AFRL-IF-RS-TR-2004-145
Final Technical Report
June 2004

PLAN AUTHORING BASED ON SKETCHES, ADVICE AND TEMPLATES

SRI International

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J860

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-145 has been reviewed and is approved for publication

APPROVED: /s/

JAMES F. REILLY
Project Engineer

FOR THE DIRECTOR: /s/

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2004	Final Apr 00 – Dec 03

4. TITLE AND SUBTITLE
PLAN AUTHORING BASED ON SKETCHES, ADVICE AND TEMPLATES

5. FUNDING NUMBERS
C - F30602-00-C-0058
PE - 63760E
PR - ATEM
TA - P0
WU - 10

6. AUTHOR(S)
Karen L. Myers, Peter A. Jarvis, Thomas J. Lee,
W. Mabry Tyson, and Michael J. Wolverton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SRI International
333 Ravenswood Avenue
Menlo Park California 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency AFRL/IFSF
3701 North Fairfax Drive 525 Brooks Road
Arlington Virginia 22203-1714 Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-145

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: James F. Reilly/IFSF/(315) 330-3333/ James.Reilly@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

Artificial intelligence (AI) planning technology provides powerful tools for solving problems that require the coordination of actions in the pursuit of specified goals. To date, however, there has been limited success in transitioning this technology to significant applications in the commercial, military, and space sectors. A major obstacle to technology transfer lies with the lack of control available to potential users of planning systems. Al planning systems have traditionally been designed to operate as black boxes that take a description of a domain and a set of goals and automatically synthesize a plan for achieving the goals.

Many potential consumers of planning technology require more user-centric tools that are designed to augment human skills rather than replace them. Both the military and space communities are showing tremendous interest in user-centric planning technology that combines plan authoring and automated decision aids. The main focus of this project was the development of the PASSAT system (Plan Authoring based on Sketches, Advice, and Templates), a mixed-initiative framework for developing complex, hierarchical plans. With its combination of interactive and automated capabilities, PASSAT enables a user to quickly develop plans that draw upon past experience encoded in templates but that are customized to individual preferences and the demands of the current situation.

14. SUBJECT TERMS
Active Templates, Artificial Intelligence, PASSAT, Planning, Decision Aides

15. NUMBER OF PAGES
99

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1. Introduction

Artificial intelligence (AI) planning technology provides powerful tools for solving problems that require the coordination of actions in the pursuit of specified goals. To date, however, there has been limited success in transitioning this technology to significant applications in the commercial, military, and space sectors. A major obstacle to technology transfer lies with the lack of control available to potential users of planning systems. AI planning systems have traditionally been designed to operate as *black boxes*: they take a description of a domain and a set of goals and automatically synthesize a plan for achieving the goals. Human planners, however, are generally reluctant to cede full control to automated planning systems in this manner.

Many potential consumers of planning technology require more *user-centric* tools that are designed to augment human skills rather than replace them. This observation has led, in recent years, to the development of a number of *plan-authoring* frameworks. Plan-authoring systems provide a set of plan editing and manipulation capabilities that support users in developing plans. These systems introduce a degree of structure to the planning process, yielding principled representations of plans with well-defined semantics. Plan-authoring systems can include a range of planning aids that reason over this structure; however, the role of such automated aids is to augment human planning skills by facilitating human-driven plan development. Both the military and space communities are showing tremendous interest in user-centric planning technology that combines plan authoring and automated decision aids because of the potential to improve the quality and process of plan development without incurring the high knowledge modeling costs and loss of control associated with fully automated planning systems.

We pursued three lines of work on this project within the general theme of user-centric planning technologies designed to augment human planning skills.

A. PASSAT

The main focus of the project was the development of the PASSAT system (Plan Authoring based on Sketches, Advice, and Templates), a mixed-initiative framework for developing complex, hierarchical plans. At its heart, PASSAT is a plan-authoring system in which users construct and modify plans interactively. Users can draw upon a library of *templates*, to the extent they desire, to assist with plan development. Templates correspond to a form of hierarchical task network (HTN), and may encode both parameterized standard operating procedures and cases corresponding to actual or notional plans developed for related tasks.

To complement these interactive tools, PASSAT includes a range of automated and mixed-initiative planning capabilities. Users can invoke an *automated planning* mode based on standard HTN methods to expand open tasks within a plan. A mixed-initiative *plan sketch* facility helps users refine outlines for plans to complete solutions, by detecting problems and proposing possible fixes. *Advice* enables users to define high-level policies for plan content that the system enforces during interactive and automated plan development. PASSAT also includes *process facilitation* mechanisms to aid the user in managing incomplete tasks during plan development.

With its combination of interactive and automated capabilities, PASSAT enables a user to quickly develop plans that draw upon past experience encoded in templates but that are customized to his individual preferences and the demands of the current situation.

B. CODA

PASSAT provides technology to support a single user in the development of a complex plan. To complement that capability, we developed a framework for supporting a team of human planners who must work together to create a common global plan. In a jumpstart effort to this project (funded under DARPA Contract No. F30602-97-C-0067), we developed an initial prototype system called CODA (Coordination of Distributed Activities) that addresses the problem of multiplanner coordination. On this contract, we generalized and extended this framework to provide a transition-ready system for use within the Special Operations Forces (SOF) community.

C. Qualitative Reasoning about Plans

One of the most important benefits from developing a structured representation of a plan is the ability to perform causal analysis to determine the impact of changes on the plan. The current standard for reasoning about plan causality focuses on linking enabling effects of actions or initial world conditions with preconditions of subsequent actions. This approach is ill-suited to interactive planning environments in which users may be constructing ad hoc plans on the fly. For this reason, we developed a qualitative approach to reasoning about plans that does not require the full causal annotations dictated by traditional methods.

Our work on this project presents a substantial departure from most work in the artificial intelligence community on planning systems through its emphasis on supporting a user in the development of complex, real-world plans. We believe that continued work in this area is essential to enable the development of technology that can assist human planners faced with increasingly complex planning tasks.

The remainder of this report is organized as follows. Section 2 describes the core plan authoring capabilities within PASSAT. Section 3 describes our work on robust plan sketching, while Section 4 describes our work on policies. Section 5 describes our framework for reasoning qualitatively about the effects of changes on a plan. Section 5.3 describes our framework for qualitative reasoning about plans. Section 6 summarizes our work on CODA within the context of this contract. Finally, Section 7 presents our conclusions for the project. Appendix A contains copies of the main technical publications produced on this contract.

2. Plan Authoring within PASSAT

2.1. Overview of PASSAT

PASSAT is a plan-authoring system in which users construct and modify plans interactively (Myers et al. 2002). Users can draw upon a library of *templates*, to the extent they desire, to assist with plan development. Templates correspond to a form of hierarchical task network (HTN) (Tate 1977; Erol et al. 1994), and may encode both parameterized standard operating procedures and cases corresponding to actual or notional plans developed for related tasks.

To complement these interactive tools, PASSAT includes a range of automated and mixed-initiative planning capabilities. Users can invoke an *automated planning* mode based on standard HTN methods to expand open tasks within a plan. A mixed-initiative *plan sketch* facility helps users refine outlines for plans to complete solutions, by detecting problems and proposing possible fixes. *Advice* enables users to define high-level policies for plan content that the system enforces during interactive and automated plan development. PASSAT also includes *process facilitation* mechanisms to aid the user in managing incomplete tasks during plan development. Such assistance is critical in complex applications, as it helps the user stay focused without overlooking important details.

Two key principles guided the design of PASSAT:

(a) Flexible, 'out of the box' planning: Traditional AI planning systems lock users into the set of solutions implied by a domain's predefined action models. Within PASSAT, templates are viewed as guidelines for performing tasks; the human planner is free to expand the set of solutions defined by the templates. In particular, a user can override constraints, drop tasks, or insert additional tasks to match his personal preferences or the demands of the current situation. This flexibility is critical for domains in which correct and comprehensive collections of templates cannot be provided.

(b) Controllable user-centric automation: Automated capabilities should complement human planning skills and be readily directable by a human.

2.2. Technology Scope

PASSAT is generic, domain-independent technology but is tailored toward applications for which (a) the complexity of the domain precludes full capture of all relevant planning knowledge, and (b) human input is critical, but some amount of automation would improve plan quality and reduce overall planning time. Special Operations Forces (SOF) mission planning, the driving domain for the Active Templates program, has these characteristics.

Standard operating procedures exist for many high- and mid-level activities in the SOF domain, and are readily amenable to encoding within an HTN representation. For

example, a hostage rescue operation can be characterized as consisting of the high-level objectives of performing reconnaissance in the areas around the rescue site, establishing a safe haven to which to remove the hostages, undertaking the assault to rescue the hostages, and transporting the hostages to the safe haven. Low-level operations follow standard doctrine and can also be modeled in a relatively straightforward manner.[1] Intermediate strategy decisions pose a bigger challenge. For example, informed selection of areas and methods for reconnaissance requires deep background knowledge of reconnaissance operations, breadth of understanding of the current situation, and significant experience. Capturing and modeling this type of strategic knowledge in full presents a tremendous challenge.

SOF planning lies well beyond the range of current automated planning technologies; moreover, fully automated solutions are unlikely ever to succeed because of the difficulty in formulating strategic knowledge with sufficient fidelity. In contrast, a PASSAT-style plan-authoring system provides a good technological match for the SOF planning domain. Missions arise unexpectedly, resulting in a need to assemble high-quality plans rapidly. Thus, the availability of tools to expedite plan development is important. Because many types of SOF operations can be broadly characterized with predefined templates, knowledge bases can be developed that capture certain portions of the planning process. However, individual operations tend to be highly distinctive, making it important to have tools that enable users to modify and customize plans to suit the needs of a particular situation.

Many potential application domains for planning technology share these characteristics of having partially formalizable domain knowledge and requiring significant user input to produce high-quality, situation-specific plans. On the military side, examples include air operations, disaster relief planning, and noncombatant evacuation operations. Space applications include science mission planning and ground operations planning.

2.3. PASSAT Example

Figure 1 shows a snapshot of the PASSAT interface during a planning session. The large frame on the left contains a hierarchical decomposition of the current partial plan. Items next to folder icons are tasks that have been expanded; items next to star icons are tasks that can be expanded further (either through automated template application or interactively); and items next to document icons are tasks that match no templates. The frame on the upper right shows the current *agenda* – the list of planning steps the user must perform to address outstanding issues. The frame on the lower right shows the list of *information requirements* – sources of information that have been identified by the user or PASSAT's planning knowledge as relevant to various portions of the planning process.

---

[1] Many of our templates were derived directly from SOF field manuals.

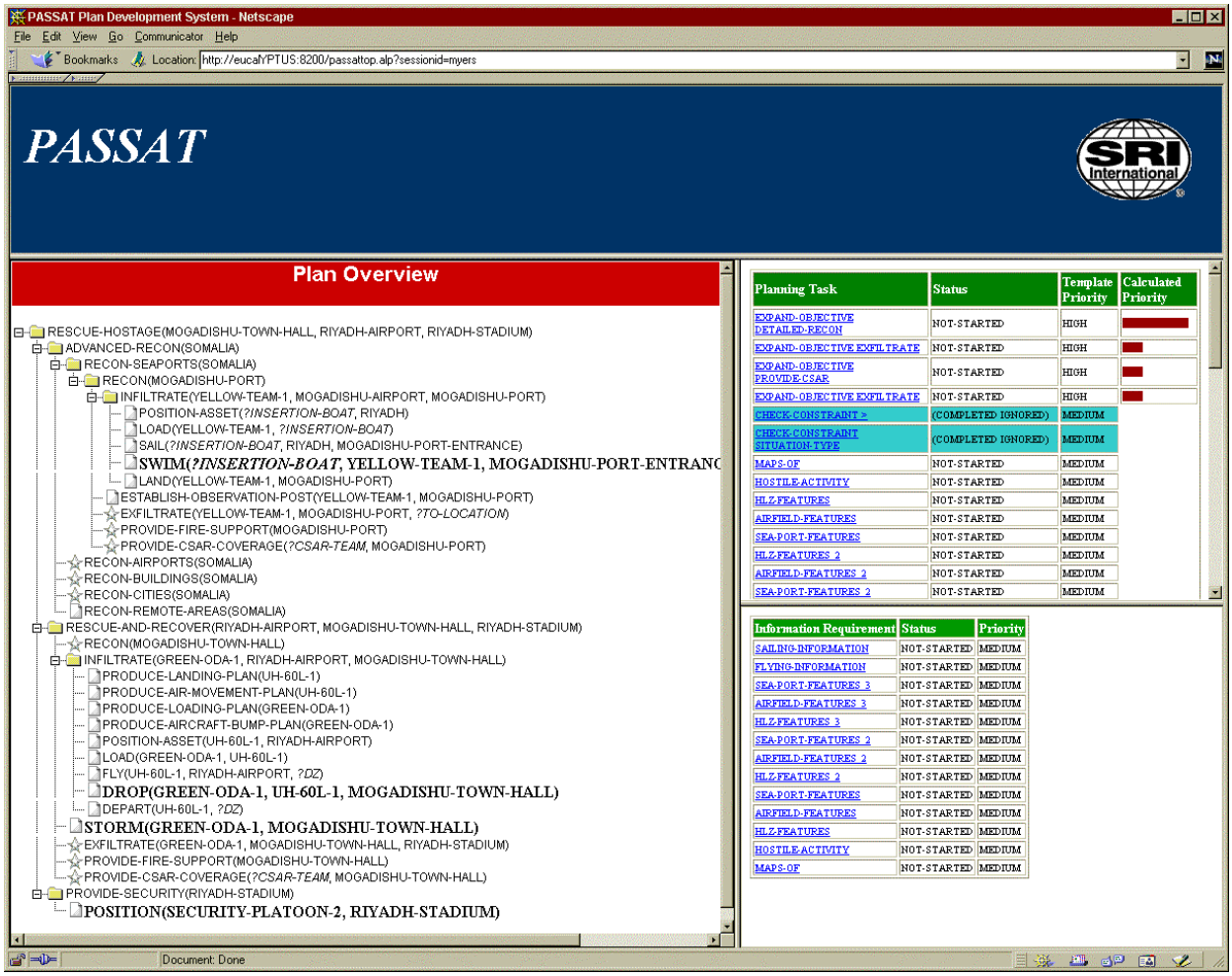

Figure 1. PASSAT Interface during Plan Development

The human planner develops the plan by selecting a planning step from the agenda and performing that step (many of these planning steps are accessible through the plan display as well). If the planning step is to expand the PROVIDE-CSAR-COVERAGE task, for example, the planner would be presented with several options: apply one of the templates that matches the task (see Figure 2), enter an expansion manually, or create a sketch for achieving the PROVIDE-CSAR-COVERAGE task and work with PASSAT to refine that sketch. Performing this planning step may cause additional planning steps to be added to the agenda (i.e., new tasks, variables, and constraints may have been introduced into the plan) and new information requirements as well.

2.4. Plan Representation

PASSAT's representation of plans and tasks is based on a fairly standard HTN model (similar to that of (Erol et al. 1994)), augmented with a rich temporal representation for tasks. Using PASSAT, a user would describe the objective of the plan in the form of one or more *task statement*s, each consisting of a *task operator* and *terms* (variables, instances, or functions applied to terms).

Templates A *template* describes one way that a task (i.e., the template's *purpose*) can be decomposed into subtasks. A template consists of a set of these *subtasks*, as well the *variables* used in the template, *constraints* on the applicability of the template, and the *effects* of successfully performing individual tasks and the entire template. Different templates may describe different decompositions for the same task.

Figure 2. A Candidate Template for Task Refinement

PASSAT's template representation supports two features not found in the framework of (Erol et al. 1994), namely, *information requirements* (discussed in detail below) and *enumeration* tasks. Enumeration tasks enable the specification of a set of tasks relative to a set of terms that satisfy a designed predicate. For example, the enumeration task

$$\forall ?city.\mathrm{DISTANCE}(?city, ?hostage\text{-}locn) < 20 \Rightarrow \mathrm{RECON}(?city)$$

indicates that a RECON task should be performed for each city within the specified distance. Other HTN frameworks (e.g., O-Plan (Currie and Tate, 1991) and SIPE-2 (Wilkins 1993)) provide similar mechanisms for enumerating subtasks relative to a designated constraint.

Constraints Constraints consist of state predicates that denote hard or soft conditions, perhaps due to physical laws or policy rules. PASSAT employs a three-valued logic for constraints, grounded in the values `TRUE`, `FALSE`, and `UNKNOWN`.

Automated constraint checking is performed when constraints are created or modified in the plan. Checking of ground constraints may return a status of `UNKNOWN`, if the information is not specified in the world state; such constraints would need to be validated explicitly by the user. Checking of nonground constraints occurs only when the number of possible instantiations is less than a predefined threshold, with the system testing whether the constraint is valid or invalid for each (i.e., establishing that the constraint is necessarily true or false independent of the instantiation). Otherwise, the system returns `UNKNOWN` and the constraint is rechecked when more variables are instantiated.

Unlike in automated planning systems, a constraint with value other than `TRUE` does not necessarily halt the process or cause backtracking. Instead, a violated constraint is called to the attention of the user, who has the choice of ignoring the violation or changing the step that triggered the violation.

Temporal Representation PASSAT supports the scheduling of tasks via constraints on the earliest and latest possible times for the start and end points of tasks. Temporal constraints typically refer to these end points but may also refer to upper and lower bounds on those time points. Temporal constraints can also be expressed using Allen's interval relations (Allen 1984).

Domain Definition PASSAT utilizes a number of coordinated databases to define its application domain. An *ontology* (based on the Generic Frame Protocol representation (Karp et al. 1995)) defines the hierarchical organization of classes and instances and their properties. *State predicate* and *task statements* are declared, specifying the number and classes of their arguments. *Functions* are similarly declared, with the additional declaration of the class of the function's value. Some predicates and functions are computable (e.g., <, +, and `Distance`) while others are defined by their extent. The *world state* is defined by a set of ground state predicates.

2.5. Mixed-initiative Plan Development

A user directs planning in PASSAT through a browser-based interface. PASSAT provides two main modes of mixed-initiative plan development: *interactive plan refinement,* and *plan sketching*. Here, we describe interactive plan refinement; plan sketching is described in Section 3.

Interactive plan refinement in PASSAT involves three types of planning step: *expand task*, *instantiate variable*, *resolve constraint*.

Expand Task For task expansion, the system offers the user the choice of applying a predefined template, specifying a set of subtasks interactively, sketching a solution (see below), or dropping the task.

When the user chooses a template to apply, the system unifies the task and the template's purpose, making appropriate substitutions throughout the template. PASSAT adds the subtasks and constraints of the template to the plan. PASSAT also extends its agenda to include planning steps to expand the new subtasks, to check the new constraints, and to instantiate any unbound variables from the template. The planning step for the parent task is marked as completed and removed from the agenda.

PASSAT checks the status of all constraints created during task expansion. For a valid constraint, the planning step to check it is removed from the agenda. For an invalid constraint, the planning step is flagged.

Other planning steps may be affected by a task expansion. If the expansion results in the assignment of a variable, the planning step for instantiating that variable is removed. Also, the status of constraints that contain that variable might now be resolvable; the system checks those constraints and updates the planning steps, if necessary.

Instantiate Variable To aid with variable instantiation, PASSAT presents to the user the set of values that satisfy all relevant constraints; the user can select from this set, provide an alternative value (hence, override a relevant constraint), or simply mark certain values as unacceptable. When the variable is instantiated, any impacted constraints are rechecked.

Resolve Constraint As noted above, PASSAT provides automated checking of constraints as part of template application, with the agenda being used to track constraints that the system was unable to validate. *Resolve constraint* steps enable a user to declare that the system can disregard designated unsatisfied constraints in a given situation. A user may wish to do so because (a) he has more recent information that would validate the constraint, (b) he knows that the constraint is overly strong for the current situation, or (c) he wants to explore a *what-if* scenario.

2.6. Temporal Visualization

The tree-oriented representation of the plan within PASSAT's browser-based interface (e.g., see Figure 1) provides a good overview of the hierarchical structure of a plan. This type of view is valuable for understanding the relationships among actions and high-level objectives. It does not, however, provide insight into the temporal structure of the plan.

Because temporal information is critical to understanding and evaluating a plan, PASSAT includes a capability to display a timeline-based view of a plan. This temporal presentation of a PASSAT plan makes use of the SOFTools Temporal Plan Editor (TPE). In particular, we developed a translator that maps a PASSAT plan (complete or partial) into the internal plan representation used by the SOFTools TPE (i.e., the .sof representation). Because the .sof language is weaker than that of PASSAT, the translated plans provide only a subset of the content of a PASSAT plan. Still, there is sufficient information to communicate the temporal nature of a PASSAT plan.

The translator exploits two data sources. First, there is a mapping of the primitive actions and their arguments within the PASSAT domain model to SOFTools graphical constructs. For example, a PASSAT `swim' action maps to a SOFTools Move arrow with an accompanying swimmer icon. The drawings of places for the SOFTools timeline are determined by exploiting type information for PASSAT actions. Second, template-specific drawing routines define aggregated display capabilities. The template-level mappings are not necessary, as each template can be reduced to its constituent actions. However, such mappings can lead to improved layouts of a plan by providing specialized directions on how to draw commonly occurring idioms.

2.7. Usability Features

We have incorporated several features into PASSAT to facilitate its use within real applications.

Because the development of a plan may span several days or be interrupted by other duties, PASSAT offers the ability to save a plan and to restart it later. As PASSAT is further developed to support multiple planners working on a single plan, this facility will allow parallel efforts to be coordinated in a shared plan repository.

A planner may sometimes develop a part of the plan and realize that the initial idea will not work. The system currently allows the user to *undo* the steps in reverse order. In the future, the user will be able to back out of earlier steps without necessarily losing later, independent steps.

PASSAT is designed to reduce the chance of inadvertent errors. Strong typing for task, function, and predicate definitions enables the checking of inputs for consistency. If a processing error should occur in the system, the undo mechanism can provide recovery to a safe checkpoint.

2.8. Modal Truth Criterion

One of the key benefits of automated planning technology is the ability to identify harmful interactions among actions within a plan.

Figure 3 shows a simple example from the Special Operations Forces (SOF) domain of a harmful interaction that involves actions that might arise during the planning of a boat exfiltration. After the execution of the first action `boat1` will be located at `green-beach`. The pickup from `red-beach` clearly cannot be executed immediately after this sailing action, as the boat will not be at `red-beach`. More formally, an effect of the first action `At(boat1 green-beach)` conflicts with a precondition of the second action `At(boat-1 red-beach)`.

The example in Figure 3 is extremely simple, as it involves two successive actions. More generally, however, interactions can occur among actions scattered throughout a plan. The Modal Truth Criterion (MTC) is a well-understood algorithm designed to detect action interference in partial-order plans (Tate 1977; Chapman 1987).

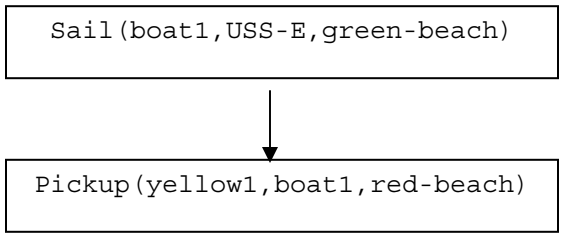

Figure 3. Conflicting Exfiltration Actions

We have implemented an MTC reasoning module in PASSAT to enable the planner to detect harmful interactions between planned actions. Our implementation is based on the standard algorithms developed by Tate (1977). It rests on top of an explicit representation of the preconditions and effects of actions within templates.

Figure 4 shows the encoding of this knowledge for the simple exfiltration example. The effects of the `Sail` action document that the boat has moved from its initial location at the `USS-E` after the action is performed, and will be located at `green-beach`. The preconditions of the pickup action require that the boat be located at `red-beach`.

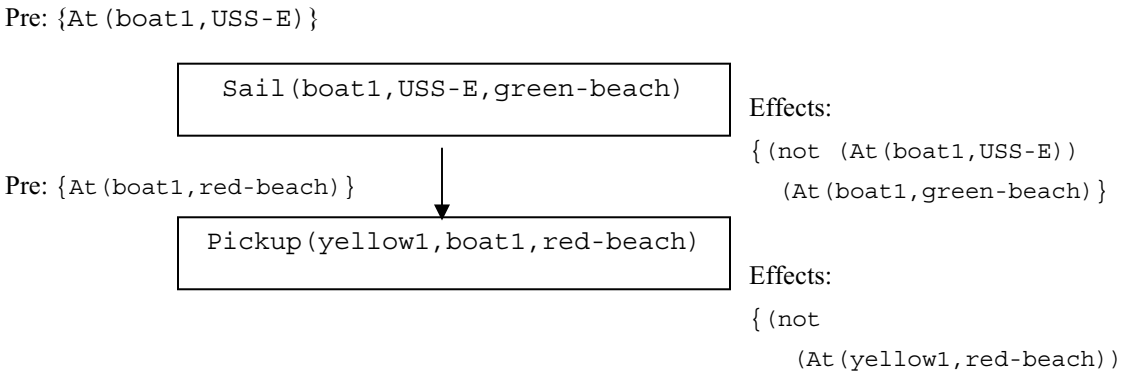

Figure 4. Example with Action Preconditions and Effects

The MTC conflict detection procedure checks every action's precondition in a plan to determine if it is satisfied or violated by the effects of other actions. In our example, the MTC reasoning module will determine that the `At(boat1,red-beach)` precondition of the pickup action is violated by the `At(boat1, green-beach)` effect of the `Sail` action.

Conflict detection is a computationally expensive task, as the system must identify for each precondition the set of actions that could potentially support and interact which occur in the closest temporal proximity to it. We have made this process as efficient as

possible by recording a look-up table that enables quick determination of the actions in a plan that are applicable in checking a given precondition.

Previous implementations of the MTC have centered on fully automated solutions where the MTC conflict detection routine is run after each plan state modification[2] (add action, bind variable, etc.). Identified conflicts are added to the planner's agenda as "flaws" that the planner needs to resolve. This model is not well-suited to a mixed-initiative planning environment since many of the conflicts that arise during plan authoring are temporary and will be resolved by further plan edits (Peot and Smith, 1993). A user will quickly become irritated with an agenda full of such transitory problems.

The challenge in a mixed-initiative framework is to identify only those conflicts that truly warrant a user's attention. Our approach takes into account the agenda of outstanding planning tasks that PASSAT maintains (see Section 2.9). We identify two threat classes for categorizing the conflicts identified by the MTC and use that categorization to control the conflicts presented to the user. A *definite threat* is a conflict between action preconditions and effects that cannot be resolved by the items currently on the plan agenda. A *possible threat* is a conflict between action preconditions for which we can identify at least one agenda item that might resolve it.

To illustrate, first consider the plan fragment in Figure 4. For this small plan, there would be no outstanding agenda items because all variables are instantiated and the actions are totally ordered. Thus there are no further steps to take in completing the plan that would eliminate the problem. As such, the interaction between the two tasks would be a definite threat. In contrast, consider the incomplete plan in Figure 5. Here, the variable `?boat` has not yet been instantiated; this type of process information is tracked by PASSAT's agenda. In this case, there is a possible threat in the case where `?boat` is later bound to `boat1`; the threat can be avoided, however, by making a different binding choice.

Within PASSAT, the user can actively browse potential threats. However, only definite threats appear on the agenda. This approach eliminates the distractions that could arise from overloading the user with many temporary problems that subsequent planning activities would correct without additional intervention by the user.

Our solution makes the assumption that we have complete knowledge about the plan agenda. This assumption does not hold in situations where a user is allowed to make arbitrary edits to a plan. Such an environment would require an extension of our definition of definite threats to include the notion of *plan publication*: a definite threat could occur only in a published portion of the plan, as we would not expect future edits to be made to those plan elements.

Pre: {`At(?boat,USS-E)`}

---

[2] The frequency of "critic" invocation is an important issue in automated planners. Some systems, such as SRI's SIPE-2 planner (Wilkins 1993), employ a "lazy" strategy where plan critics are invoked only after completion of a planning level. Doing so can delay the detection of problems but greatly reduces the overall amount of time devoted to constraint checking.

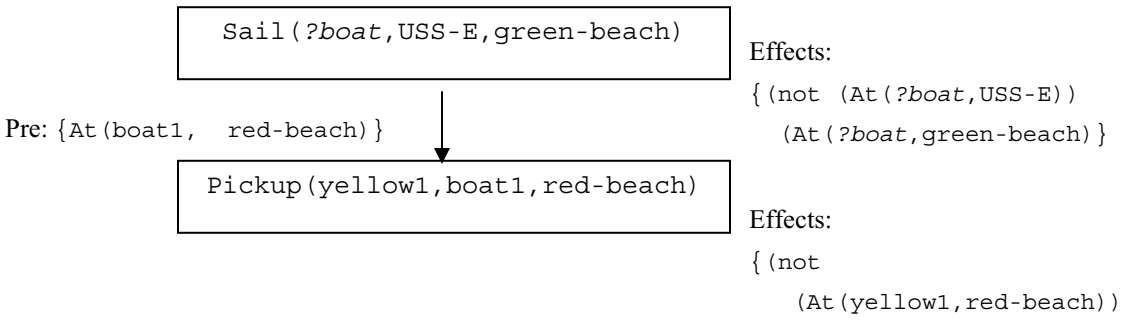

Figure 5. Example of a Possible Threat

Our work presents a first step toward a practical approach for supporting MTC reasoning within a mixed-initiative planner.. Further work is needed in the following important areas:

- **Conflict resolution support**. The key problem is the interactions between resolution options and other constraints in a plan, as selecting a resolution to one conflict may introduce new conflicts.
- **Preemptive conflict identification.** We are currently using the MTC in a reactive mode where we inform the user of the problems introduced into a plan after plan modification options have been selected and applied. It would be useful to provide a look-ahead function that could preemptively identify conflicts associated with available options so that users could avoid introducing conflicts.

2.9. Process Facilitation

PASSAT facilitates the user's plan-authoring process by helping the user track information that is important to the development of the plan. Process facilitation is supported primarily by two capabilities:

- A prioritized *agenda* of planning steps listing the decisions that the user must make to address problems or incompleteness in the current plan.
- A mechanism for identifying key *information requirements* implicit in the user's partial plan, and for directing the user's attention to relevant plan elements when new information arrives.

2.9.1. Agenda and Prioritization

PASSAT's agenda consists of the open planning steps facing the user given the current state of planning. By 'planning steps', we mean decisions and actions that the user makes in the process of developing the plan; these are distinguished from the activities that are part of the plan itself. PASSAT currently supports three types of planning step – *expand task*, *instantiate variable*, and *resolve constraint* – described earlier. The planning

steps PASSAT displays in its agenda can be filtered by the user along several dimensions, including step type and completion status. The user can also sort the agenda along several dimensions, including step type, creation time, and alphabetical order. The filtering and sorting facilities can be especially useful for helping the user find a particular step on the agenda.

In real domains, the development of a plan can involve hundreds or even thousands of decisions. Correspondingly, PASSAT's agenda can grow quite long during the planning process. The system provides some basic mechanisms to control agenda growth – instantiating variables during template application, automatic calculation of constraints – and to control information overload in the agenda display – the aforementioned agenda filtering and sorting. However, even with these capabilities, the agenda can frequently reach a size that is overwhelming to the user. In the face of a large number of planning steps, we need a technique for keeping the human planner focused on the most important ones.

To deal with this problem, we have developed mechanisms for prioritizing the planning steps on the agenda, according to some notion of a step's importance to the planning process. Our approach has been to offer a suite of prioritization tools, from which the user may choose given the specific planning situation. PASSAT supports three prioritization approaches:

- **Predefined** Each subtask, variable, and constraint in a template may be tagged with a qualitative priority (*high*, *medium*, or *low*), corresponding to the importance of making a decision about that entity (expanding the task, instantiating the variable, checking the constraint). Predefined priorities always take precedence over PASSAT's other prioritization methods in ordering the agenda display.
- **Commitment-based** This approach prioritizes each planning step according to the degree that a decision will constrain the rest of the planning process, giving highest priority to the most constraining decisions. This criterion is especially useful in collaborative planning situations, where it is important to make decisions early when they will constrain the alternatives available to other planners. Our technique measures commitment as the expected number of future decisions eliminated by performing the step. We approximate this with a recursive formula that performs a lookahead search through the plan space. While we use some simple heuristics to reduce the size of the search, the current procedure is still reasonably expensive relative to PASSAT's other update calculations. As a result, the current implementation of commitment-based prioritization covers only tasks. In future work, we will investigate techniques for approximating the commitment level of a planning step more efficiently.
- **Experience-based** In contrast to the commitment-based approach, which is an attempt to identify what the planner should do next based on some theoretical model of planning, the experience-based approach bases its prioritization on what

real human planners have done first in the past. The experience-based prioritization technique stores preference histories of planning steps, and learns a preference function for them using the online learning algorithm of (Cohen et al. 1998). Planning steps are indexed by the step type, the object name, and the 'call stack' of templates that created the object.

Other possible methods for deriving a step's priority include

- *Urgency-based:* prefer decisions that involve execution tasks that are scheduled to start soon.
- *Backtracking-based:* prefer decisions that are difficult to achieve. This is effectively the prioritization criterion of the Fewest Alternatives First strategy and related heuristics (Pollack et al. 1997) used in automated planning.
- *Depth-first:* prefer steps that derive from the steps most recently performed by the user. This approach assumes that the user wants to remain focused on one area of the plan before moving to another.
- *Breadth-first:* prefer steps that derive from the steps least recently performed by the user.

2.9.2. Information Requirements

In real-world planning, the human planner often makes decisions based on criteria that are too complex or vague to formalize in a predicate. These criteria are often based on external sources of information (e.g., reports, meetings). For example, a SOF planner may want to base his selection of a rendezvous point on an overall assessment of an intelligence report from the relevant region, though it may be virtually impossible to formalize the exact set of conditions the planner is looking for within that report. In a plan-authoring system, we want to be able to capture these criteria and information sources, and record the connection between them and the relevant elements of the plan. PASSAT accomplishes this through the use of *information requirements*.

In addition to specifying the method for expanding a task, a template may also include one or more information requirements. An information requirement specifies a monitoring condition on an information source that may be useful for determining the applicability of the template, for selecting variable instantiations, or for resolving the template's constraints.

Currently, information requirements are used in PASSAT to make explicit to the user the connection between plan elements (e.g., *variables*, *constraints*) and information sources. When a planner activates an information requirement in a template, the system creates a link between the information described in the information requirement and an element or elements in the plan. When the information arrives, PASSAT calls the planner's attention to the relevant plan element by creating a high-priority item on the agenda to revisit that element. PASSAT's current method of detecting when information has arrived is to be told explicitly by a user, but one could imagine more sophisticated automated *sentinels*

that would, for example, monitor data sources (e.g., Web pages, databases) for specific updates.

For example, a user planning a SOF mission may make a tentative assignment to a variable `?RENDEZ-POINT` based on the sketchy information available to him. At the same time, he may activate an information requirement representing an intelligence report on the region in question and attach it to the variable `?RENDEZ-POINT`. When the intelligence report comes in, PASSAT will notify the planner by putting the Instantiate Variable step for `?RENDEZ-POINT` back on the active agenda, giving it a high priority, and highlighting the element on the planner's agenda display.

2.10. PASSAT System

The PASSAT system can be run on either a Windows or Sun workstation platform. There is an extensive user guide for the system that includes (a) instructions for downloading and installing PASSAT, (b) a detailed overview of the PASSAT interface, and (c) extended demonstrations of the core plan authoring and sketch processing capabilities. The demonstrations make use of a hostage rescue domain that was developed on the project. The domain consists of an extensive set of templates that encode standard operating procedures related to hostage rescue, a background ontology of generic object classes and instance information for a specific scenario, and a small library of predefined advice.

3. Robust Plan Sketching

3.1. Overview

Automated hierarchical planning systems support a top-down model of planning focused on the refinement of high-level objectives to executable actions. Human planners, in contrast, often combine top-down planning with a bottom-up approach that identifies specific tasks to be included in a final solution. Indeed, studies have shown that designers tend to interleave decisions at various levels of abstraction, thus working opportunistically at times rather than in a purely top-down fashion (Guindon 1990). For example the planners of a hostage rescue may decide where and how they will establish a safe haven and how hostages will be transported, without yet having selected an overall rescue strategy. The selection of high-level strategy, in fact, can often be conditioned on such lower-level decisions.

Within PASSAT, we developed a framework that enables a user to *sketch* an outline of a plan for a particular objective, with the system providing assistance in refining the outline to a full solution (Myers et al. 2003). A sketch consists of a collection of tasks that (1) may be only partially specified, and (2) may occur at various levels of abstraction in the plan hierarchy.[3] Within this framework, a human planner can combine opportunistic and top-down plan refinement in a manner that best suits his individual planning style.

We had previously developed an approach to plan sketching that required plan sketches be *valid*, meaning that there be at least one legal completion of the sketch relative to predefined planning knowledge (Myers 1997). Mismatches between human conceptualizations of a domain and formalized planning knowledge, however, can lead to situations where user sketches are uninterpretable. On this project, we addressed the problem of plan sketch interpretation when the validity assumption no longer holds.

Below we summarize our main contributions in this area, namely, the definition of concepts and algorithms for interpreting and repairing invalid plan sketches in a robust manner, and the implementation of a framework for mixed-initiative sketch repair.

3.2. Interpreting Invalid Sketches

Our earlier work on plan sketching introduced a notion of *plan sketch compliance*, which is based on the idea of embedding the tasks of a plan sketch within an overall solution to the given planning problem. In particular, each sketch task must be unifiable with some corresponding task in the hierarchical plan structure that constitutes the solution. Robust plan sketching requires a less stringent condition that can account for both (a) user misconceptions about the task domain (i.e., situations where the user has incorrect models of when and how activities can be undertaken), and (b) background knowledge

[3] Sketching often implies a graphical medium. While our model of sketching is compatible with graphical specification of tasks, we considered only logical specifications.

that may be incorrect or incomplete. We focused on two types of problem within sketches that derive from user misconceptions and faulty domain knowledge:

- *Type 1:* violations of constraints from the templates used to interpret a plan sketch
- *Type 2:* sketch tasks that do not map to any high-level goal (i.e., *orphaned* tasks)

To accommodate such problems, we defined the weaker notion of *maximal compliance* to accommodate these problem types. In contrast to the original model of compliance, maximal compliance captures the notion of embedding a maximal subset of the original sketch within a plan refinement structure while minimizing constraint violations. In this way, it characterizes the class of solutions to a planning problem that best reflect a given sketch, subject to the constraints of the background knowledge.

Ideally, a robust sketch interpretation algorithm should aim to identify one or more plan refinement structures that are maximally compliant. To this end, we defined an algorithm (see (Myers et al. 2003) for details) that could be used to generate maximally compliant solutions for a given sketch. However, domain complexity may preclude finding optimal solutions in practice.

3.3. Mixed-initiative Repair

Our framework for sketch interpretation and repair could be operationalized as a fully automated system. Doing so would require powerful heuristics to control search, however, as the overall space of possible repairs will generally be prohibitively large. Our interests lay more with user-centric planning aids, which led us to develop a mixed-initiative realization of the robust sketch progressing algorithm within PASSAT.

When given a sketch, PASSAT generates possible *expansions*, which amount to least-commitment plan structures that embed the sketch (or some subset of it) and all derived consequences. The user may choose any of these expansions to continue planning; the agenda will be updated to reflect the derived set of outstanding tasks. PASSAT guides the human planner through the process of modifying a plan sketch to eliminate detected problems. The role of the system is to identify sketch problems and possible repairs, while the human acts as the decision maker in navigating through the space of options. The framework is designed for iterative use, with a human planner incrementally refining a sketch in response to detected problems until he has found a satisfactory solution.

Mixed-initiative systems require powerful and flexible interfaces to facilitate interactions with a user. To support mixed-initiative sketch repair, we developed two interactive tools: a *sketch editor* and a *sketch space exploration aid.*

Sketch Editor

Sketch specification involves defining the tasks that comprise a sketch and their arguments. PASSAT provides an interactive editor to simplify this process. With this editor, the user first selects a set of tasks to be included in the sketch, and then specifies the arguments for those tasks. Allowed arguments consist of variables and all instances of

the corresponding type for that argument. This type of structured plan editor eliminates the possibility of syntactic mistakes (e.g., undefined tasks or arguments, use of inappropriate argument types) that can be a source of great frustration to a user. In doing so, it allows the user to focus on the conceptual design for a sketch.

To help the user focus on semantically relevant choices, the sketch editor incorporates context-sensitive presentation of the syntactically valid options to the user for both task and argument selection.

- *Task selection:* The editor exploits linkage among templates to limit task selection for a sketch to tasks that could possibly appear in any expansion of the 'objective' currently under consideration. This filtering helps to eliminate many irrelevant options, thus both reducing clutter from the task selection menu and preventing the user from pursuing many fruitless avenues.
- *Argument selection:* It is often the case that many candidate values for a task argument fail to satisfy the preconditions of any templates that could be applied to the task. Eliminating such values from consideration prevents exploration of deadends. However, one design requirement for PASSAT was the flexibility to let a user think 'out of the box'. In particular, PASSAT's constraint reasoning allows certain constraints to be overridden at the user's discretion. For this reason, the possible values presented to the user are flagged to indicate whether or not they satisfy all associated constraints.

Sketch Space Exploration Tool

The space of possible expansions for a given sketch can be dauntingly large, especially when interpretation is tolerant of the Types 1 and 2 faults defined above. To support a user in navigating this large space, we have developed a *sketch space exploration* tool that aids a user in managing the sketch refinement process (see Figure 6). The tool is organized around a tree structure that reflects the space of sketches and expansions that a user has explored. The root of the tree corresponds to the initial sketch; it contains a descendant node for each expansion of the sketch. Each revision of an expansion in turn generates a descendant sketch node, from which a recursive structure emerges.

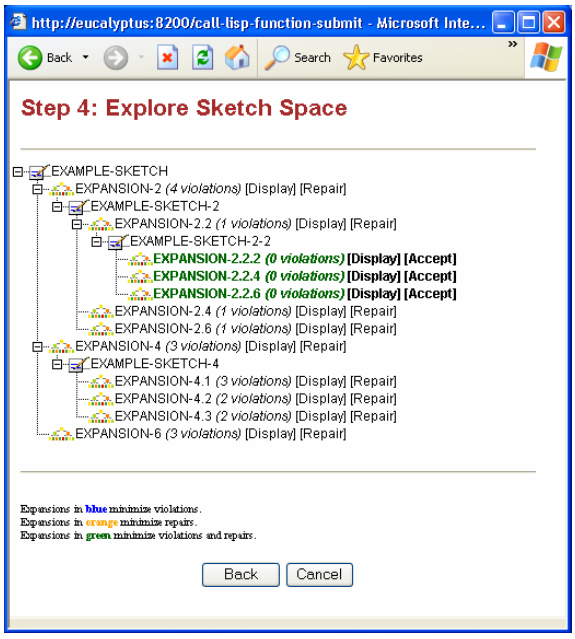

Figure 6. Sketch Exploration Tool

For a sketch node, the user can choose to generate expansions all at once or incrementally. For an expansion, a user can view the template structure and the detected

problems. Expansions with minimal problems and minimal numbers of expected repairs to address those problems are highlighted. (One repair could fix multiple problems; thus, these values can differ for a given expansion.)

3.4. Discussion

Plan sketching provides a powerful mechanism for developing complex plans. Plan sketching can help a user quickly outline the key aspects of the plan, capitalizing on the system to fill in less important details around the sketch. In addition, it can serve as the basis for an exploratory process that allows a user to consider a variety of options when developing a plan.

For plan sketching technology to be useful in practice, it is imperative that it be robust both to flaws in the sketch itself and to inadequacies in the underlying planning knowledge used to interpret the sketch. Our work has defined an approach to robust plan sketch interpretation that accommodates two categories of problem: violated applicability conditions and extraneous actions. This approach has been embodied within a mixed-initiative plan sketching framework in which a system identifies options for repair while a user selects candidate interpretations and repairs.

We believe that our work presents substantial progress toward a practical sketch-processing facility. However, there are still areas for future work. One key topic for exploration is the design of summarization and comparison tools for sketches and expansions, which would help a user better understand the structure of the sketch space. A second area is to define techniques for generating qualitatively distinct expansions that provide the user with some sense of the range of possible options for a given planning problem. Finally, it would be useful to broaden the definition of a sketch to include temporal information. Doing so would require extensions to the sketch interpretation algorithms to match partial orders of tasks; currently, the matching is done separately for individual sketch tasks.

4. Policies

4.1. Overview

When developing a plan, it is often necessary to take into account high-level guidance that is distinct from the actual objectives to be attained. Examples of such guidance include rules of engagement (e.g., *There should be no helicopter-based infiltrations within 1 mile of a school, hospital, or church*), commander's intent (e.g., *The infiltration mission should involve no more than 2 Green units*), or general strategic requirements that an individual planner wants to impose on a given plan (e.g., *Don't use more than 3 landing zones*). We use the term *policy* to refer to guidance of this type.

We developed a module within PASSAT to support the use of policies during mixed-initiative plan development. Our accomplishments included the design of an appropriate language for the representation of policies, and the development of an incremental planning-time policy verification tool linked to PASSAT's plan authoring capabilities.

Verification of policies involves monitoring the evolving plan content and user edits to identify operations that would lead to violations. Violations could be triggered either directly by a user modification (e.g., template application, variable instantiation) or indirectly through system constraint propagation or sketch processing. Within PASSAT, both actual and potential violations (for changes under consideration by the user) lead to notification. In addition, the system agenda is updated to track any actual policy violations. As with other forms of constraint violations within PASSAT, the user can choose to accept such violations, thus making policies relaxable under user discretion.

Plan policies within PASSAT are similar in nature to the concept of advice for automated planning systems that we developed in our previous work on the Advisable Planner (Myers, 1996; Myers, 2000). In particular, both are a form of declarative guidance designed to shape plan content. Advice within the Advisable Planner was designed to influence the operation of a fully automated planning system. PASSAT policies, in contrast, serve as guidance for both user and system activities (hence, the use of the term *policy* rather than *advice*). This mixed-initiative model of planning led to a validation-oriented model for policies, in contrast to the enforcement-oriented model within the Advisable Planner.

Although our policy work in PASSAT leverages concepts from the Advisable Planner, this differing treatment of the guidance required new control regimes for testing advice that are sensitive to usability issues. We also introduced advances in the policy framework that extend and improve the advice language from the Advisable Planner. One key innovation was the articulation of *aggregate policies*, which reference the entire plan structure rather than being limited to the local problem-solving context. A second innovation was a generalized *domain metatheory*, which is used as the basis for specifying policies.

The remainder of this section introduces our domain metatheory, summarizes the types of plan policy developed within PASSAT, describes our policy validation techniques, and summarizes the impact of the work.

4.2. Domain Metatheory

As with advice in the Advisable Planner, our language for representing policies builds on three components: the underlying *domain theory*, a *domain metatheory*, and the connectives of first-order logic.

A standard domain theory for an agent consists of four basic types of element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *tasks* to be achieved, and *templates* that describe available means for achieving tasks. A domain metatheory defines semantic properties for domain theory objects. These properties can be used to express preferences among otherwise equivalent options; they also enable description of activity at a level that abstracts from the details of an agent's internal representations. As discussed in (Myers 2000), a metatheory can provide a powerful basis for supporting user communication. The main concepts within our metatheory for plan policies are *features* and *roles* defined for agent plans and goals (similar to those of (Myers 1996)).

A *template feature* designates an intrinsic characteristic of a template that distinguishes it from other templates that could be applied to the same task. For example, among templates for route determination, there may be one that is `OPTIMAL` but `SLOW` with a second that is `HEURISTIC` but `FAST`; each of these attributes could be modeled as a feature. Although the two templates are functionally equivalent, their intrinsic characteristics differ significantly. Features provide the means to distinguish among such operationally equivalent alternatives.

A *template role* describes a capacity in which a domain object is used within a template; it maps to an individual variable within a template. For instance, a route determination plan may contain variables `location.1` and `location.2`, with the former corresponding to the `START` and the latter the `DESTINATION`. Roles provide a semantic basis for describing the use of individuals within templates that abstracts from the details of specific variable names. Roles also provide the means to reference a collection of semantically linked variables that span different templates (i.e., `START` roles may occur in multiple templates).

The metatheory within the Advisable Planner was limited to features and roles defined for plan templates. Within PASSAT, we further support the definition of features and roles for tasks. This extension simplifies the process of defining a metatheory for a domain. In particular, roles and features can be specified 'one time' as part of task definitions, rather than being repeated with each occurrence of the task within a planning template. It turns out to be valuable, however, to retain the ability to specify roles and features on templates; doing so enables the use of context-specific features and roles that are specialized to the particular problem-solving strategy embodied within a given template.

4.3. Types of Plan Policy

We support two types of policy within PASSAT: *role* and *aggregate*.

4.3.1. Role Policies

A role policy either prescribes or restricts the use of domain entities for filling certain capacities in the plan. Role policies are characterized by the schema:

```
<Use/Don't Use>  <objects> in <roles> for <context-activity>
```

In general, a role policy consists of one or more roles, a role-fill constraint, a context activity, and a polarity indicating whether the policy is prescribing or prohibiting the role-fill. For example:

Only choose infiltration landing zones that are within 5 miles of the infiltration target

Here, the context activity is defined as tasks with feature *`Infiltration`*. The policy dictates that the fillers for the roles *`Landing-Zone`* and *`Infiltration-target`* be within 5 miles of each other.

4.3.2. Aggregate Policies

Aggregate policies are defined in terms of six values: *role, role-constraint, attribute, aggregation-function, test-relation,* and *test-value*.

Aggregation is with respect to the set of fillers for *role* that satisfy the designated *role-constraint*. An aggregation value is produced by first selecting the designated *attribute* for each object (if one is specified) or the object itself (if no value is specified). The *aggregation-function* is applied to this set to define an aggregation value. The policy prescribes that this aggregation value must satisfy the *test-relation* relative to the defined *test-value*.

For example, the aggregate policy "*Use no more than 1 Green-Unit as an infiltration team"* would be defined as

```
Role: Infiltration-team
Role-constraint: ((IS-TYPE INFILTRATION-TEAM GREEN-UNIT)
Attribute:     none
Aggregation-function: SIZE
Test-relation: <
Test-value:    1
```

Aggregate policies are valuable for expressing restrictions that amount to a form of global measure for a plan. Such measures often involve limits on resource usage, time bounds, and plan size.

4.4. Policy Validation

Our approach to validation for role policies builds substantially on previous algorithms from the Advisable Planner. However, our implementation was notably different because

of the need to accommodate mixed-initiative planning (in contrast, the Advisable Planner's focus is on providing guidance to direct a fully automated planner). In particular, we introduced a 'preprocessing phase' that performs preliminary checking of policies when the user is considering possible options for extending a plan (i.e., either template application or variable instantiation). This proactive checking identifies possible policy violations that might arise for each of the choices under consideration by the user, thus enabling a user to avoid triggering a violation and then later having to take steps to undo it.

Aggregate policies represent a technical innovation over our previous work. One key technical challenge that they introduce is the requirement for a mechanism to efficiently track global plan changes. We implemented an incremental mechanism for policy verification that can track the status of aggregate policies by monitoring individual plan changes as they occur, thus avoiding expensive global rechecking of aggregate policies for the entire plan.

The use of task-based roles and features (as opposed to just template based) also required modifications over the original Advisable Planner algorithms. In particular, consideration of a particular template for application needs to take into account the new subtasks and their metatheoretic properties when checking policies.

4.5. Summary

Plan policies provide a powerful mechanism for defining, tracking, and enforcing high-level guidance that should shape the content and form of a plan. They provide the means by which to readily incorporate both external constraints on the plan (e.g., commander's intent, rules of engagement) and to help an individual planner enforce his own high-level design criteria.

Our contributions in this area consist of the definition of a powerful language for expressing policies that are relevant to shaping plan content and the design of verification techniques that are linked into the mixed-initiative plan authoring framework. Together, these provide the basis for an effective policy framework.

We see three areas for future work on plan policies:

- Role and aggregate policies represent two useful types of policy but others are possible. For example, the Advisable Planner supports 'method advice', which provides a way to express recommendations on types of templates to apply in designated contexts (e.g., use a *covert* (rather than *noncovert*) operation). For a particular domain, it would be useful to work with a set of experts to determine what would constitute a comprehensive policy language that covered their full set of requirements.
- Our policy verification capability within PASSAT assumes that all policies are defined at the start of a planning session. Ideally, users would be able to specify policies throughout the planning process and have them checked relative to the current plan.

- We did not develop a tool for specifying policies as part of our work on this project. Instead, users of PASSAT must select from a library of predefined policies. Predefined policies make sense for applications where there would be a limited set of higher-level constraints that may be imposed on the planning process. In general, however, a user should have the ability to define situation-specific policies as the need arises. Technologies such as Constable (http://www.isi.edu/ikcap/constable/publications.html), another Active Templates effort, provide suitable platforms on which to build this type of policy specification tool.

5. Qualitative Causal Reasoning

Our work on qualitative causal reasoning focused on developing a practical, user-oriented framework for reasoning about plans that does not require comprehensive background models for possible actions. In particular, we sought to provide mechanisms for answering the following important questions:

A. What role does a given action, constraint, or assumption play in a plan?

B. What impact would a given change have on a plan?

To this end, we defined a *qualitative* approach to reasoning about plan structure that builds on (a) a set of qualitative and casual-link plan relations that characterize key interactions among plan components, and (b) an accompanying calculus for reasoning qualitatively about the effects of changes on a plan. As will be seen below, our qualitative approach trades the precision of traditional causal link methods for simplicity and ease of use.

Here, we describe the motivation for this work and the core relations within our model. Full technical details can be found in (Myers 2003); Appendix A includes a copy of this document.

5.1. Background

Automated planning algorithms embody a theory of causality grounded in the linking of enabling effects of actions or initial world conditions with preconditions of subsequent actions (Weld 1999). While the design and algorithmic foundations for this approach are well understood, the approach has several limitations. First, the generation of causal plan structures by automated planning systems requires *comprehensive* causal models that describe for every action its *preconditions* (i.e., the conditions under which it could be applied) and its *postconditions* (i.e., the conditions that result from execution of the action). Second, the models must be completely *correct* in order to support the causal reasoning required for plan generation. Even the smallest of errors or omissions can lead to invalid plans, or the inability to generate any plan.

Because of these strict requirements, the standard approach to causal reasoning within AI planning systems is ill suited to interactive planning environments in which users may be constructing ad hoc plans containing actions that may not have extensive background theories. Imposing the requirement that a human planner specify a full causal structure for his plan would impose a tremendous documentation burden that most users are unlikely to satisfy, due to both a lack of modeling expertise, and an unwillingness to invest the time required to justify the low-level details of his choices.

To illustrate, consider the simple plan fragment in Figure 7, which is drawn from a domain focused on noncombatant evacuation operation (NEO) missions. The figure presents the three actions in the plan (in bold) along with a full causal annotation of the type required for causal link reasoning within an AI planning system. These annotations explicitly note any required preconditions above the action and postconditions below the

action. Actions and effects are represented in terms of an action/effect name followed by a list of parameters of the form *?param-name = param-value*, indicating the parameters for the action/effect and their bindings. Arrows denote causal links from enabling conditions to required preconditions.

Clearly, the fully annotated view of the plan contains much more information than would be practical for a human planner to provide for plans that contain hundreds or thousands of actions (which is typical of real-world domains such as military planning, crisis action planning, and space mission planning). Although such information is necessary for an automated planner, much of this information would be of little value to a human planner because it is obvious from the plan structure (e.g., knowing that a LOAD operation results in the cargo then being LOADED on the chosen vehicle is a form of bookkeeping that is unlikely to interest the human planner). On the other hand, certain of these relationships may be important to understanding key dependencies within the plan, or could encode critical conditions whose monitoring is essential to ensure plan viability in the face of unexpected changes.

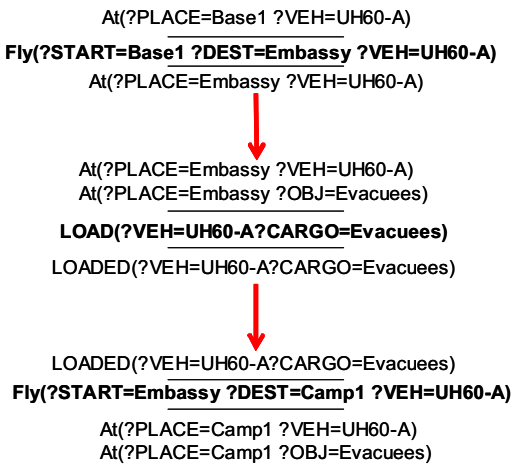

Figure 7. Evacuation Plan Fragment with Full Casual-Link Annotations

5.2. Qualitative Reasoning about Plans

Are traditional causal-link models necessary for meaningful analysis of plans? We believe the answer to be *no*. More specifically, we believe that much (but not all) of the value of these complex causal models can be obtained through simpler *qualitative* models that capture commonsense notions of intraplan relationships. The basic idea is to trade the detail and precision of the formal causal models for an approach that is both easier to formulate and to reason with. In particular, a human planner would be able to specify such relations as part of the plan authoring process. This simpler approach would still enable answers to the questions (A) and (B) above, although in qualitative rather than purely logical terms. Furthermore, as we argue below, the models of causality embraced by the AI planning community are unnecessarily narrow, because of their evolution from

the structures required for fully automated planning. In particular, they do not capture notions of plan dependency that are important for mixed-initiative planning systems.

On this project, we developed a specific technical approach that supports this type of qualitative plan reasoning. We defined a set of qualitative plan relations, as well as a calculus for reasoning qualitatively about the effects of plan changes that considers both qualitative and standard logical plan relations. By covering both types of relation, the calculus can both capitalize on traditional logical models of causality when available, and also produce meaningful results for partial, qualitative models of plan relations.

5.3. Plan Relations for Qualitative Reasoning

In this section, we present a candidate set of relations to support qualitative reasoning about the effects of plan changes. Ideally, a system that reasons about causal effects within plans should combine both causal link and qualitative information about plan relationships. For this reason, our model contains relations of both types.

We adopt a model of plans that contains three types of element:

Action (denoted by *A*): an activity that can be undertaken

Effect (denoted by *E*): a condition (either to be achieved, the expected result of executing an action, or a property of the initial world state)

Parameter (denoted by *P*): an argument to an action or condition

We use the symbol *Obj* (i.e., plan object) to denote an arbitrary plan element from any of the above types.

A plan relation is defined between a *source object* and a *target object.* We represent a qualitative relation using the syntax:

Source-Obj → *Target-Obj*

5.3.1. Causal Link Relation

The *causal-link* relation (Figure 8), the standard relation within most automated planning systems, indicates that the target effect is dependent on the source effect.

Causal-link: {Effect} → *{ Effect }*

Figure 8. Casual Link Relation

5.3.2. Qualitative Relations

Figure 9 summarizes our candidate set of qualitative relations. Broadly speaking, the qualitative relations can be separated into two categories: *temporal* (QR1 – QR3) and *logical* (QR4 – QR5).

QualR1. *Precedes: {Action | Effect} → {Action | Effect}*

QualR2. *Necessary-for: {Action | Effect} → {Action | Effect}*

QualR3. *Supports: {Action | Effect} → {Action | Effect}*

QualR4. *Parameter-dependence: {Parameter} → {Parameter}*

QualR5. *Condition-dependence: {Effect} → {Action | Effect | Parameter}*

Figure 9. Qualitative Plan Relations

5.3.2.1. Qualitative Temporal Relations

The qualitative temporal relations capture the notion that a given action or effect in a plan must precede some other action or effect, possibly due to some semantic relationship between the elements. We consider three types of temporal relation: *precedes*, *necessary-for*, and *supports*.

The *precedes* relation captures the notion that the specified source action or effect should take place before the specified target action or effect, without providing any indication of why. This type of relation can be used to capture a preference for performing activities in some designated order when there is no necessary reason for that order. For example, consider the actions of preparing an evacuation site and flying evacuees to the evacuation site. Although it would be possible to perform those actions in parallel, a given planner may have a preference for completing the preparation prior to the start of the airlift of the evacuees, possibly to enable a delay of the airlift in the event of problems with the preparation.

The *necessary-for* and *supports* relations specialize the *precedes* relation to capture semantic motivations for the ordering relationship. *Necessary-for* captures the notion that a given action or effect must occur before a designated action or effect in order to enable the target plan element. For example, it would be *necessary-for* evacuees to be marshaled to an assembly point before they could be loaded onto an evacuation aircraft. In essence, the *necessary-for* relation constitutes a qualitative counterpart of the quantitative *causal-link* relation. Changes could impact plan objects linked by a *necessary-for* relationship in two ways. First, delays to necessary activities will propagate. Second, failure of a task that is *necessary-for* another task would likely jeopardize the latter.

The *supports* relation indicates that the source action or effect contributes to the target action or effect in some way, but is not critical to its existence. For example, a CAP mission may support a given evacuation activity, but not be essential to its undertaking Hence, if the fighter involved with the CAP were redirected to support a different action, the evacuation process should not be jeopardized. Source objects for *supports* relations correspond to 'redundant' actions or effects that, while unnecessary, lead to improved plan robustness or quality. Standard causal link models do not explicitly support such redundant objects within a plan.

5.3.2.2. Qualitative Logical Relations

The qualitative logical relations QR4 and QR5 capture the idea that there is some sort of dependency between the source and target elements such that a change to the source could impact the target. However, the precise nature of that relationship is not captured algebraically in terms of a deductive specification or mathematical formula. Such situations arise frequently in planning situations, where many factors that are problematic to formalize come into play when making choices. These factors could include conditions that are too complex to codify (i.e., a form of the *qualification problem*) or subjective preferences that vary among different human planners.

For example, the choice of assembly point in an evacuation plan will necessarily impact the type of aircraft that can be used for transporting evacuees (e.g., a small helicopter may be necessary for evacuation from an embassy, while a larger aircraft could be used at a football stadium). However, there is no hard-and-fast rule for determining what type of aircraft should be used for a particular location. Similarly, the security level of the surrounding area may constrain the choice of assembly point, but generally will not uniquely determine it.

Qualitative logical relations can be designated between plan parameters (QR4), or between plan effects and any type of plan component (QR5). The relationship between the choice of assembly location and transport aircraft in the example above corresponds to a *parameter-dependence* relation, while the relationship between the security level and choice of assembly point corresponds to a *condition-dependence* relation.

We note that the qualitative logical relations could be made 'quantitative' by associating definite constraints with them. For the *parameter-dependence* relation, these constraints would be in the form of a set of equations linking the two parameters. We introduce the term *parameter-constraint* relation to refer to this specialization of the *parameter-dependence* relation. A comparable *condition-constraint* relation could also be defined.

5.4. Example

To illustrate the use of the qualitative relations for planning, consider the plans in Figure 10 and Figure 11 for a simple evacuation operation. These plans are designed to achieve the two goal conditions *Prepared(Camp1)* and *At(Evacuees Camp1)* based on the operators in Figure 12. Each of the plans includes conditions from the initial state upon which actions and effects in the plan depend, as well as the effects that constitute the desired goal state. To simplify reference, each action in the plan is labeled with a unique identifier (e.g., *N1*).

The plan in Figure 10 corresponds to a solution that an automated causal link planner might produce for this problem. It includes a full causal link annotation that documents how action precondition is supported by an earlier effect in the plan.

The plan in Figure 11 represents a solution that a human planner might construct using some kind of plan authoring tool. It contains all of the actions in Figure 10 plus one additional action: a *Patrol* action (*N7*) that provides additional security for the sector to

which the evacuees will be moved. According to the logic of the domain operators, this action is redundant because it does not establish any effects that are required within the plan. However, it is typical for human planners to build such redundancy into plans to provide additional safeguards in the face of unexpected events.

The plan in Figure 11 also contains a candidate set of both quantitative and qualitative relations that document what the user might view as the key dependencies within the plan. The key differences between this hybrid set of plan relations and the causal link relations in Figure 10 are as follows:

- Causal-link relations in Figure 11 have been limited to dependencies on initial state conditions that might be expected to change and hence may require modifications to the plan (e.g., the security of key locations and the position of the vehicle to be used for transporting the evacuees) and important intermediate effects of action (e.g., the evacuees remain at the embassy until they are loaded onto the transport vehicle). In particular, static initial conditions and unimportant intermediate effects of actions have been omitted.
- The hybrid annotation replaces certain of the quantitative *causal-link* relations with qualitative *necessary-for* relations, indicating that it is essential for the source activity to precede the target activity in the plan but not documenting the effects that link the actions. From the user's perspective, these effects are obvious (e.g., the aircraft has to be loaded before it can be unloaded) and so documenting them explicitly is of little value.
- The qualitative annotations include a *precedes* relation from node *N6* to node *N4*, indicating a (noncausal) preference for ordering those two actions, although the ordering is not necessary for the plan to succeed.
- A *condition-dependence* relationship has been added from the predicate *#Evacuees(25)* in the initial world state to the parameter *?PLACE* in *N1* where the evacuees are to be assembled. This relation reflects the fact that the choice of assembly location is dependent on the number of evacuees; should the number change, the choice may need to be revisited.

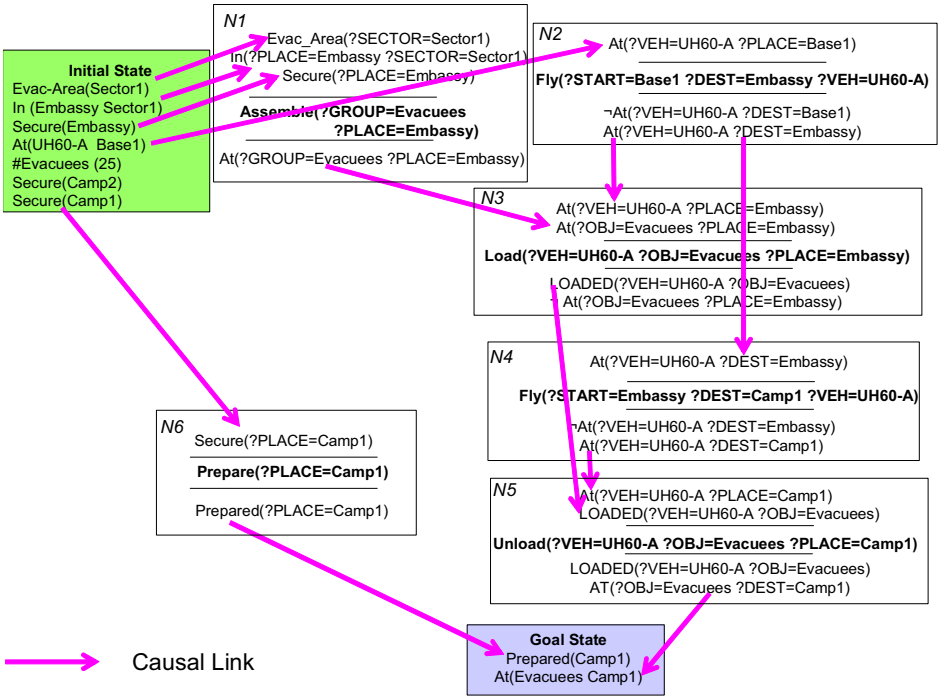

Figure 10. Evacuation Plan with a Complete Set of Casual Link Relations

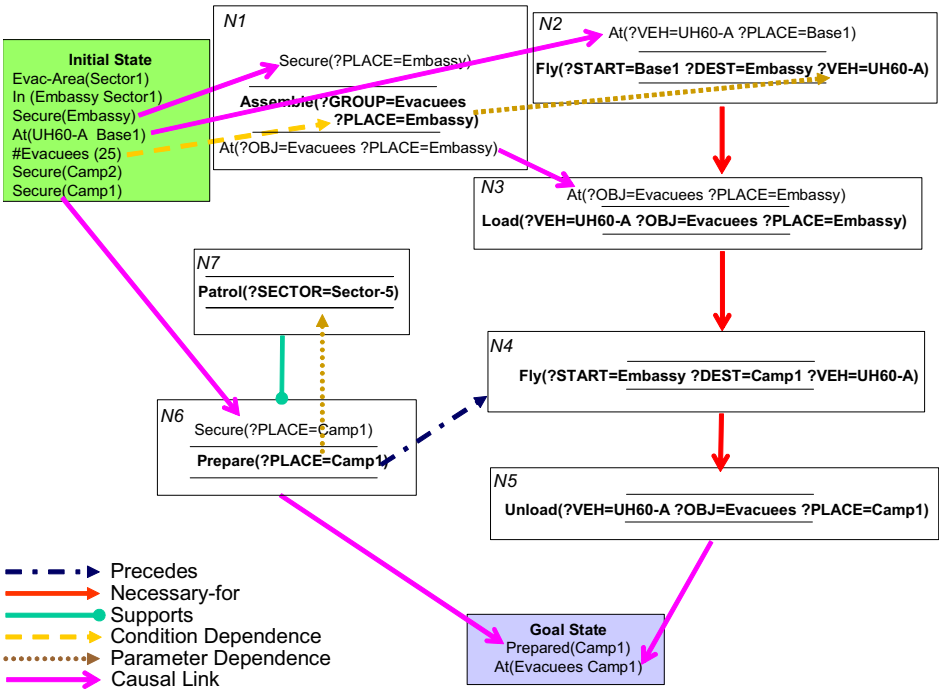

Figure 11. Evacuation Plan with Qualitative and Casual Link Relations

- A *parameter-dependence* relationship has been added from *?PLACE* in *N1* to *?Veh* in *N2* and a *condition-dependence* relation added from *#Evacuees(25)* in the initial world state to *?Veh* in *N2*. These relations show that the choice of vehicle depends on both the number of evacuees and their assembly location.
- There is a *supports* relation from *N7* to *N6* in the qualitative view, documenting that the *Patrol* action is being performed in service of the *Prepare* action. No link between these nodes is possible in the causal link view because there is no enabling relationship between effects produced by *N7* and required by *N6*. A *parameter-dependence* relation has been added from the *?PLACE* parameter in *N6* to the *?SECTOR* variable in *N7* indicating that the choice of patrol area depends on the evacuation site. In this case, the relationship could be expressed algebraically, i.e., by adding the annotation *IN(?PLACE ?SECTOR)* to the *parameter-dependence* relation (thus resulting in a *parameter-constraint* relation).

Action: *Assemble(?GROUP ?PLACE)*
Preconditions: *Secure(?PLACE), Evac-Area(?SECTOR), In(?PLACE ?SECTOR)*
Effects: *At(?GROUP ?PLACE)*

Action: *Fly (?START ?DEST ?VEH)*
Preconditions: *At(?VEH ?START)*
Effects: *At(?VEH ?DEST), ¬At(?VEH ?START)*

Action: *Load(?VEH ?OBJ ?PLACE)*
Preconditions: *At(?VEH ?PLACE), At(?OBJ ?PLACE)*
Effects: *Loaded(?VEH ?OBJ), ¬At(?OBJ ?PLACE)*

Action: *Unload(?VEH ?OBJ ?PLACE)*
Preconditions: *At(?VEH ?PLACE), Loaded(?VEH ?OBJ)*
Effects: *At(?OBJ ?PLACE), ¬Loaded(?VEH ?OBJ)*

Action: *Prepare(?PLACE)*
Preconditions: *Secure(?PLACE)*
Effects: *Prepared(?PLACE)*

Action: *Patrol(?PLACE)*
Effects: *Prepared (?PLACE)*

Figure 12. Evacuation Planning Operators

5.5. Properties of the Model

The qualitative model trades the precision of exhaustive causal links for simplicity and ease of use. Indeed, there is a natural abstraction from an exhaustive causal link structure to 'corresponding' qualitative models that involves replacing every *causal-link* relation with a *precedes* or *necessary-for* relation. We refer to a plan transformed in this manner as a 'qualitative abstraction' of the original.

While the qualitative relations lose precision over the causal approach, they offer several advantages. First, they are simpler and more intuitive to specify. This characteristic makes them better suited for use in a mixed-initiative planning environment. A second advantage relates to expressivity: there are relationships that can be modeled in the qualitative framework that are not supported by POCL-style causal links or parameter constraints. In particular, the *supports* relation enables the description of a connection between plan objects that is not essential for plan correctness (as described above). As well, the *precedence* relation allows the expression of ordering information independent of causality. Furthermore, the *condition-dependence* and *parameter-dependence* relations enable ill-defined connections between plan objects to be expressed; this ability is useful when the precise logical or mathematical relationship is not known or not easily formalizable, yet there is still a desire to document some relationship between them.

5.6. Sources for Causal Relations

Our motive for investigating qualitative relations is to enable reasoning about plan changes within a mixed-initiative planning environment that combines human and automated planning skills. Within this context, several sources would contribute to the set of causal relations defined within a given plan. First, a background set of templates could capture both quantitative and qualitative causal relations for 'standard operating procedures' that arise frequently in practice. Such relations would be included in the core template definition by the author of the template. Second, the human planner that is driving a specific planning task could contribute additional relations. Third, some relations could be derived from a general-purpose background theory that describes general properties of a domain. Finally, some sort of learning mechanism could be applied to hypothesize qualitative relations from a user-authored plan, yielding a baseline that a user could then modify appropriately (e.g., along the lines of (El Fattah and Dyer 2001)).

5.7. Summary

Our qualitative model trades the precision of POCL-style causal links for simplicity and ease of use. In particular, qualitative reasoning about the effects of changes on plans has several advantages over standard logical/deductive approaches. First, qualitative reasoning does not require comprehensive and correct causal theories. However, while qualitative inferences can be drawn from incomplete models, more complete models will yield more informative results. Second, qualitative relations are simpler and more

intuitive to define, making it possible for users to annotate plans with qualitative relations that reflect their specific needs and interests. In contrast, traditional deductive approaches require sophisticated models that have proven to be difficult for users to formulate. Third, qualitative models include relationships that do not require complete formalization of concepts, making them relevant for situations where precise dependencies among plan elements cannot be articulated.

One of the problems with the qualitative framework described above is that it is derived from simple commonsense notions of plan relations and interactions. Future work in this area should focus on defining a formal semantic model that relates the qualitative relations to POCL-style plan annotations. Such a model would enable us to ground the intuitions behind the qualitative relations in well-understood, clear structures.

6. CODA: Coordination of Distributed Activities

Effective coordination of distributed human planners requires timely communication of relevant information to ensure overall coherence of activities and the compatibility of assumptions. The CODA system (Coordination of Distributed Activities) provides directed information dissemination as a way of improving coordination among distributed human planners (Myers, Jarvis and Lee 2002).

An initial prototype CODA system was developed on a jump-start effort to this project (supported by DARPA Contract No. F30602-97-C-0067). Our work on CODA for this contract focused on the following enhancements to that original framework.

- Extension of the preliminary CODA proof-of-concept system into a transitionable system
- Integration of CODA with appropriate tools from the Active Templates program
- Development of a set of complementary interfaces for PAR specification
- Publication of two technical papers describing the CODA framework

Following a brief overview of the CODA technical approach, details of the first three of these areas are provided below.

Fred Bobbitt, Warren Knouff, and Kelly Snapp provided excellent feedback to us during our CODA development efforts on this contract, especially with regard to user requirements and interface design. We are grateful for their assistance.

6.1. CODA Approach

CODA targets applications where distributed human planners are assigned responsibility for portions of a global shared plan. Their individual subplans, while somewhat independent, are expected to have a medium degree of coupling through the need to reflect coherent strategy, to coordinate actions, and to share limited resources.

Within CODA, an individual planner declares interest in different types of plan changes that could impact his local plan development. These declarations, called *plan awareness requirements* (or *PARs*), would be registered with all other planners who are contributing components to the overall plan. Each individual user develops his plan with a CODA-compatible plan authoring system, publishing the results to a central plan server when his plan is mature enough to be reviewed by other planners. A user can initiate the checking of the PARs that he has registered at any time, and will receive notification of matches to plan components that have been published to the central plan server. CODA could be linked to a range of manual, semi-automated, and fully automated planning tools. In this project, it was connected to a specific plan editor, the SOFTools Temporal Planner (GTE, 2000), which supports the graphical editing of special operations plans.

CODA supports two classes of PARs.

1. *Plan-state* PARs describe conditions of a plan and are modeled in terms of a well-formed formula in the plan query language. For example: *There is an arrival to staging base Gold scheduled for after 8 PM.* Matching of a plan-state PAR occurs when a modification results in a plan that satisfies the associated plan query.
2. *Plan transition* PARs describe changes between two plan states. We distinguish several categories, based on the nature of the underlying plan changes:
    - **Instance Creation PARs** are used to declare interest in the addition of an object to a plan that satisfies stated conditions. For example: *Addition of decision points related to weather calls.*
    - **Instance Deletion PARs** are used to declare interest in the removal of an object from a plan that satisfies stated conditions. For example: *Elimination of a landing zone south of the embassy.*
    - **Instance Modification PARs** are used to declare interest in the modification of an object that satisfies stated conditions. For example: *Changes to movements by the 4th platoon.*
    - **Attribute Modification PARs** specialize Instance Modification PARs to changes to a specific attribute of a plan object, possibly satisfying stated change conditions. For example: *Delays of > 1 hour in the expected time to secure the church.*
    - A**ggregate Modification PARs** can be used to declare interest in changes to an intensionally defined collection of objects. The change may be to membership in the collection, or to some *aggregation value* defined over the collection. As an example: *Decrease of > 2 in the number of fire-support aircraft.*

Figure 13 presents the CODA architecture. Within the context of a global plan, individuals work independently to produce local plans for their assigned tasks. Plans are developed using a structured *plan editor*, which supports a broad range of plan manipulation capabilities. User interactions with the plan editor are tracked by an *observer* module, which maintains a complete history of editing operations. As events are logged, a semantically grounded representation of the local plan is built within CODA. This internal representation can be annotated and used for reasoning, independent of the plan editor GUI.

The *matcher* provides the main inferential capability within CODA, being responsible for linking observed plan changes to declared PARs. The matching process may involve reasoning with a background theory, whose role is to bridge the gap between low-level plan edits and PARs expressed in high-level languages. When matches are detected, notification is sent to the local planner who registered the matched plan awareness requirement.

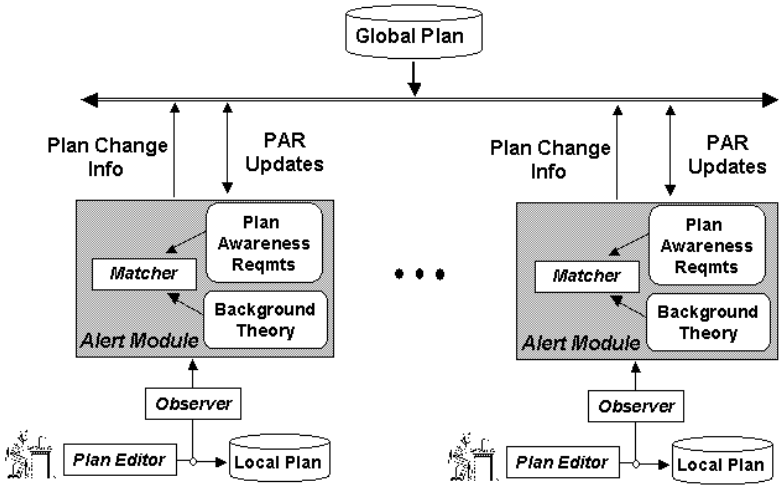

Figure 13. CODA Architecture

6.2. From Proof of Concept to Transition-Ready System

The CODA system developed on the jumpstart project (version 0.5) was designed as a proof-of-concept system to demonstrate the potential of the PAR-based approach to information dissemination among distributed planners. The initial system was limited in scope, being focused primarily on demonstrating the declaration and matching of PARs. One objective on this contract was to develop the CODA system into a mature system that could be transitioned to the SOF community. Here, we summarize the tasks that we performed to meet that objective.

6.2.1. Matching Modes

The original CODA system provided *immediate* notification of PAR matches. In particular, PARs were checked after every plan edit operation, thus providing planners with real-time notification of relevant plan changes. Immediate notification of this type would be suitable for the end stages of planning (when plans are mostly stable and changes are significant), or during execution.

For earlier stages of plan development, frequent and wide-ranging changes to plans would be expected; real-time notification of matches during early plan development would be counterproductive. For this reason, we developed a second mode of matching called *on-demand*, for which matching information is provided only in response to an explicit user request. Such requests produce summaries of matches for the current plan relative to a designated 'checkpoint' plan. On-demand matching can support coordination of distributed planners earlier in the planning process by enabling a given planner to periodically check for changes by other planners that could impact his own efforts. The checking process for on-demand matching need not consider the intervening plan modifications, since the match semantics compare the original and current plans.

6.2.2. Distributed Architecture

Within the original proof-of-concept system, a single user played the role of both a local planner engaged in planning and a remote planner who registers PARs. That initial design was chosen to expedite development of a basic demonstration system but clearly was not adequate. On this project, we implemented a distributed version of the system that allows multiple users of CODA to coordinate on plan development across geographically distributed areas.

Our distributed framework relies on two technologies developed outside of SRI. First, it uses the Structured Data Model (SDM) from GDAIS (via the DB Proxy tool) to serve as a centralized plan repository. Second, it employs the TIM router from ISX for asynchronous message exchange among CODA modules.

Within this framework, individual planners 'publish' significant versions of their local plans to the SDM, which are then accessible to all CODA agents. CODA agents read plans from the SDM to initialize their ontologies of plan objects. Checking of PARs is initiated under user control (i.e., CODA's real-time matching has been deactivated).

6.2.3. Linkage to Operational SOFTools

CODA Version 0.5 was linked to the research version of SOFTools (version 1.2d). In particular, CODA relied on special hooks within that version of SOFTools to provide notification of plan changes that could impact PAR matching.

In the final year of the Active Templates project, we migrated CODA from version 1.2d of SOFTools (the research version) to the operational version being developed by GDAIS (starting with version 2.0, currently with version 3.0).

CODA was designed originally around an 'event-oriented' model, in which individual edits prosecuted by a user were tracked. This tracking would then enable reasoning to detect changes that match user-specified PARs. This event-oriented approach enables real-time notification of relevant plan changes, but requires modifications to the plan development tool to track individual plan edits. Because of the operational focus of the GDAIS team, we were unable to obtain the necessary hooks within SOFTools 3.0 to support such event monitoring. For this reason, we migrated CODA to a 'plan-oriented' model, in which matching is done relative to incremental versions of published plans. As part of this work, we developed an XML parser that reads .SOF versions of a plan and stores them into CODA's internal plan representation.

Our update to run with the operational version of SOFTools introduced a key limitation into CODA. The nonresearch versions of the SOFTools systems include a number of attributes for plan entities that are entered manually by a human planner as free text. We had modified earlier versions of SOFTools to replace textual entries with selections from a predefined ontology. This ontological grounding of attributes enables CODA to perform more sophisticated reasoning about plans than is possible with unconstrained text strings. Within our current version of CODA, we have opted to treat text fields within the nonresearch SOFTools systems as 'interpretable', making the assumption that

user-supplied text corresponds to a predefined ontology element. This assumption enables us to demonstrate the power of CODA within the limitations of the nonresearch SOFTools systems, until such time as it incorporates anontology for plan terms. However, it does render the system nonrobust with respect to inputs that are not defined without our internal ontology.

6.2.4. Session Management

CODA supports a model of asynchronous collaboration, in which users can drop in or out of a distributed planning session and still receive the benefits of CODA services. In particular, the CODA module for a new user is initialized automatically to include the currently registered PARs of all other active users. Other session-related capabilities include notification of when users log on or off, and automatic deregistration of PARs on logoff.

6.2.5. PAR Extensions

We extended the underlying PAR representation in two ways to meet the requirements of real-world applications. First, we added priorities to PARs. Our priority model consists of the levels of importance. One important use of priorities is to enable different types of notification services based on PAR significance. Within the current CODA system, for example, notifications are displayed in red, green, or blue, depending on the priority level of the matched PAR. These priorities could also be associated with different communication modalities (e.g., notifications of high-priority matches are sent by beeper, while others are sent by email). Second, we added a notion of context to PARs organized around individual operations plans, and missions. This contextualization enables an operator, who may be engaged in planning for a number of operations simultaneously, to localize his information updates to the specific planning contexts where they are relevant.

6.3. PAR Specification Tools

The initial CODA prototype relied exclusively on a programmatic interface for specifying PARs. While adequate for internal use, this interface would be inappropriate for operational personnel. On this contract, we developed a range of complementary PAR specification tools to support various types of CODA users and their needs, namely

- a forms-based editor that covers the full range of the PAR language
- an object-based interface for quick specification of a limited range of PARs
- a library mechanism that supports user selection from predefined collections of PARs

During a particular planning session, we would expect a planner to draw on libraries of predefined PARs primarily to form the basis of his registered interests, augmenting them as necessary with authored PARs tied to the current situation and plans.

We briefly describe each of these specification tools below.

6.3.1. Forms-based PAR Authoring

The forms-filling editor for creating PARs provides a sophisticated specification tool aimed at advanced users. This tool was developed using Adaptive Forms (Frank and Szekely 1998), a grammar-based framework from USC/ISI that supports the specification of structured data through a form-filling interface that adapts in response to user inputs. With this tool, users create PARs by filling in forms with an English-like syntax; as users incrementally specify PARs, remaining options change in accord with the constraints of the underlying grammar. An internal compiler transforms these high-level specifications into the formal PAR structures required by CODA's matcher.

Creation of an Adaptive Forms application involves specifying a grammar that defines the forms to be presented to the user. For CODA, we developed a grammar (and hence an interface) to support the full range of expressiveness of our PAR representation language (with minor exceptions).

In designing a specification tool of this type, the competing requirements of expressivity and ease of use must be balanced. Sufficient expressivity is required to ensure coverage of relevant cases; however, support for full expressivity can lead to complex and unintuitive interfaces. To address this issue, CODA's PAR authoring tool provides two sets of forms. First, a set of general forms provides the full expressive power of the PAR language, including the ability to construct arbitrary expressions in first-order logic. While powerful, these forms require more effort to complete; in addition, people unaccustomed to formal languages require training to use them effectively.

For this reason, the tool also includes specializations of the general forms that capture common idioms within the SOF planning domain. These specialized forms build in values that users would have to specify in the general case, thus simplifying and shortening the specification process. Parameters within the forms enable customization to a given planning session. SOF planners, for example, are generally interested in delays to activities. The SOF application of CODA includes (among others) the following parameterized PAR idiom:

- *Delays to any actions of greater than*
- *Delays to action*

The first form supports declaration of interest in delays to actions that exceed a duration to be supplied by the user. The second form supports declaration of interest in delays to a user-specified action within a plan. Users can create PARs based on these specialized forms simply by supplying the designated parameters.

6.3.2. PAR Libraries

The CODA library facility allows PARs to be defined ahead of time and grouped into modules according to functionality or expected usage. For example, our SOF demonstration system includes separate libraries for fire-support, maneuver, combat search and rescue (CSAR), and so on. Predefinition makes sense for many applications,

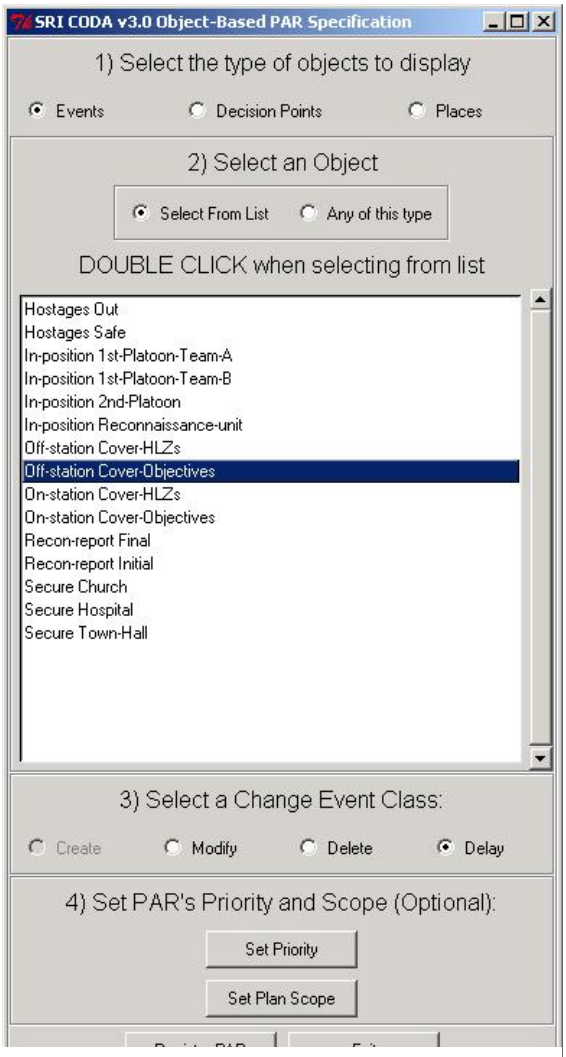

Figure 14. Object-based PAR Specification Tool

as there will often be a general set of changes that individual planners would want to monitor. For example, a team responsible for medical needs during an evacuation would almost always be interested in changes to the size of the force and the expected number of evacuees. PARs for these types of change can be stored in a library focused on the needs of medical planners.

6.3.3. Object-based PAR Specification Tool

The object-based PAR specification tool enables a user to quickly define simple PARs. It organizes the specification process around individual and classes of SOFTools objects. The user begins by selecting a class of such objects, and then selecting either a generic object of that class or a specific individual defined in some current plan. For that selection, the user specifies a mode of change in which he is interested, one of modification, deletion, delay, or creation (depending on the object type). In addition, the user can specify a scope for the change (in terms of the plan, operation, or mission to consider), and a PAR priority.

The object-based PAR specification tool, shown in Figure 14, provides an effective mechanism for quickly creating a broad range of simple but useful PARs. Its expressiveness, however, is limited. For example, while it is easy to specify a PAR expressing interest in "any delay to the Secure Church objective", a specific delay length cannot be designated.

6.4. CODA System

The CODA system runs on PC hardware under the Microsoft Windows family of operating systems (95, 98, 2000, NT, XP). There is an extensive user guide for the system that includes (a) instructions for downloading and installation of CODA, (b) a detailed overview of the CODA interface, (c) a quick-start guide to walk users through a

simple demonstration of CODA, and (d) an extended demonstration designed to capture intended use of CODA in a realistic SOF planning operation.

The demonstrations are grounded in a scenario developed by subject matter experts within the DARPA program "Small Unit Operations: Situational Awareness System". The mission involves neutralizing an enemy force that has taken position in a small town, and evacuating the town's civilian population. The basic maneuver plan includes the infiltration of reconnaissance and assault forces, the assault, and the reconstitution and exfiltration.

6.5. Future Work

We see several directions for extending our work on CODA.

6.5.1. Impact Analysis

Focused information dissemination provides one important tool for coordination. One obvious next step for CODA would be to develop tools that could help a user understand the ramifications within his local plan of matches to PARs.

That type of impact analysis requires the availability of a deep causal model for the plan. For example, suppose that a planner makes the decision to change the assembly point for an evacuation operation from the local embassy to a nearby football stadium. Such a change could impact strategy (i.e., it may now be possible to use larger helicopters for the evacuation), timing (i.e., the stadium may be further from the center of town), and security requirements. Traditional AI planning techniques have assumed that a plan includes sufficient causal information so that all consequences could be computed directly when changes occur. However, plans authored by human operators are unlikely to include that level of detail. Our work on qualitative causal reasoning (Section 5) could be used to overcome this limitation.

6.5.2. Generalized Notification Services

CODA currently supports a simple mechanism for informing users of plan changes that match their declared PARs, namely, the display of a notification message in the CODA interface. It would be valuable to consider a suite of multimodal notification methods, whose use could be customized to an individual planner. The methods could include notification via email, phone, fax, or messaging to a personal digital assistant. This broader range of notification schemes is imperative to support asynchronous models of planning, given that a user may not be 'online' with his planning system when critical changes are made of which he needs to be aware.

The notification services would build on a *user profile* that provides both contact information and preferences for an individual user. Preferences could be based on criteria such as priorities associated with PARs, user location, and the priority or ownership of the modified plans. For example, a user might simply wish to receive an email message with a summary of changes to contingency plans that triggered matches to PARs, but prefer instant notification via a cell phone or PDA if some assets under his control are to

be reassigned. The phase of the planning cycle could also bear on the choice of contact medium: for example, as H-Hour approaches, users might prefer to be paged in order to maximize the time available to consider responses.

6.5.3. Resolution Services

The basic matching services within CODA enable awareness by a local planner of changes that could impact his planning process. In situations where changes either introduce opportunities for synergy among planners (e.g., shared use of a tanker) or raise potential conflicts (e.g., the possibility of friendly fire incidents), human planners will need to collaborate to determine appropriate adjustments to their plans to reconcile the effects of the reported changes.

One version of CODA incorporated a 'chat' facility to enable users to communicate informally to respond to plan changes. The use of this medium for resolving problems is appropriate in certain circumstances (e.g., obvious modifications that are not expected to trigger controversy). However, when changes are contentious or when there are users who have gone offline, alternative communication methods are required to facilitate the resolution process.

As a next step, it would be interesting to extended CODA to include a structured proposal/counterproposal capability that could be used by human planners to negotiate modifications to plans. This facility would be grounded in an ontology of plan change proposals (e.g., *delay an activity, use an alternative resource*), with a user selecting one or more such proposals and instantiating them for the current situation. Counterproposals could be made by modifying earlier proposals, or suggesting completely new alternatives in response to the original change event.

7. Conclusions

Challenging planning problems are an integral part of many commercial, military and space endeavors. Given the number and complexity of these problems, there is a clear need to develop technology to improve problem solving in these areas. The SOF domain, the driving application domain for the Active Templates program, exemplifies a planning task for which technology could play several important roles: simplifying and expediting the planning process, enabling better solutions, and improving plan execution through support for execution monitoring and plan revision.

To date, there has been limited success in transitioning AI technology into planning applications. The root cause of the failure has been the misguided focus within the AI planning community on developing fully automated systems that could replace human decision makers. In particular, the problems to be solved in these domains far exceed the technical capabilities of current planning technology. Furthermore, the knowledge bases that would be required to capture even the basic factors that impact decision making for these domains would be difficult to create and problematic to maintain. Finally, few potential consumers of planning technology are interested in black-box solutions, as users would like to be able to shape and influence the decision-making process by which solutions are generated.

We believe that AI planning technology can and will make a difference in real-world planning problems. However, the successful technologies will not be fully automated. Rather, they will be *user-centric* technologies that provide assistance to the human planner rather than attempting to replace him. Our work on this contract has focused on these types of user-centric planning technologies, covering both the needs of an individual planner and teams of distributed planners who must work collaboratively to develop a shared plan.

The core of our work focused on the PASSAT plan-authoring system. With its combination of interactive plan authoring, plan sketching, and advice, PASSAT enables a user to quickly develop plans that draw upon past experience encoded in templates but that are customized to his individual preferences and the current situation. The human remains the key decision maker within PASSAT, but can invoke automation when appropriate to aid with task expansion, constraint checking, and process management. This style of mixed-initiative planning is essential for many domains, where the generation of high-quality, trusted solutions requires substantial human insight and judgment.

While there have been a number of mixed-initiative planning systems developed previously, those efforts have required that the system be endowed with complete and correct knowledge bases for the application domain. Within PASSAT, we sought to relax this requirement. Users can override system knowledge of constraints, when desired, during plan development. Similarly, our robust plan sketching capability can identify differences between a user's outline for a plan and what the planning knowledge within the system supports as possible, as well as mechanisms to resolve those

differences. Our approach accommodates two categories of problem: violated applicability conditions and extraneous actions.

This tolerance of incorrectness within the background models used by the planning system constitutes a first step toward a much more flexible style of mixed-initiative planning that can support 'out of the box' ideas by the user. Our work on qualitative reasoning about plans moves even further in this direction, by supporting the ability to reason about plans using a less formal characterization of the plan's structure than is required by current automated planning systems. In particular, our qualitative framework supports a level of reasoning that is commensurate with the amount of casual structure and degree of precision that a human planner is willing to provide. An approach of this type will be essential for supporting human planners as they develop complex plans that necessarily transcend predefined knowledge bases.

With the CODA framework, we developed a practical solution to the problem of coordinating the activities of distributed human planners engaged with plan authoring tools. By having human planners explicitly declare those aspects of the overall planning process that interest them, CODA enables timely and focused distribution of information that can expedite and improve the quality of coordinated problem solving. The use of a rich, AI-based representation for describing plans, planning changes, and background theories provides the key to this technology. We believe that the coordination capabilities that CODA provides can improve significantly the effectiveness of a planning team. As such, we were disappointed not to have the opportunity to transition the CODA system into operational use by the SOF community. We believe that this technology is ready for use in real-world planning problems, and will look to transition it in the context of future DARPA programs.

8. Bibliography

Allen, J. F. "Towards a General Theory of Action and Time". *Artificial Intelligence* 23, 1984.

Chapman. D. Planning for Conjunctive Goals, *Artificial Intelligence*, Volume 32, 1987.

Cohen, W. W., Schapire, R. E., and Singer, Y. "Learning to Order Things". In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds). *Advances in Neural Information Processing Systems*, The MIT Press, 1988.

Currie, K., and Tate, A. "O-Plan: The Open Planning Architecture". *Artificial Intelligence*, 32(1), 1991.

El Fattah, Y. and Dyer, D. "Causal Consistency for Plan Authoring and Execution: A Preliminary Report". Unpublished Manuscript, 2001.

Erol, K., Hendler, J., and Nau, D. (1994). Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Frank, M.R. and Szekely, P. "Adaptive Forms: An Interaction Paradigm for Entering Structured Data". In *Proceedings of the ACM International Conference on Intelligent User Interfaces*, 1998.

GTE, *SOFTools User Manual*, 2000.

Guindon, R. "Designing the Design Process: Exploiting Opportunistic Thoughts". *Human-Computer Interaction*, 5(2), 1990.

Karp, P. D., Myers, K. L., and Gruber, T. "The Generic Frame Protocol". *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1995.

Myers, K. L. "Strategic Advice for Hierarchical Planners". In *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th Intl. Conference* (L.C. Aiello, J. Doyle and S.C. Shapiro, eds.), Morgan Kaufmann Publishers, 1996.

Myers, K. L. "Abductive Completion of Plan Sketches". In *Proceedings of the 14th National Conference on Artificial Intelligence*, AAAI Press, 1997.

Myers, K. L. "Planning with Conflicting Advice". In *Proceedings of the Fifth International Conference AI Planning and Scheduling (AIPS2000)*, AAAI Press, Menlo Park, CA, 2000.

Myers, K.L., Jarvis, P.A., and Lee, T.J. "Active Coordination of Distributed Human Planners". In *Proceedings of the Sixth International Conference on AI Planning Systems*, Toulouse, France, 2002.

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., and desJardins, M. "PASSAT: A User-centric Planning Framework". In *Proceedings of the 3rd*

International NASA Workshop on Planning and Scheduling for Space, Houston, TX, 2002.

Myers, K. L., Jarvis, P. Tyson, W. M., and Wolverton, M. J. "A Mixed-initiative Framework for Robust Plan Sketching". In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, Trento, Italy, 2003.

Peot. M. and Smith. D. "Threat-Removal Strategies for Partial-order Planning". In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993.

Pollack, M. E., Joslin, D., and Paolucci, M. "Flaw Selection Strategies for Partial-Order Planning". *Journal of Artificial Intelligence Research* 6, 1997.

Tate, A. "Generating Project Networks". In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.

Weld, D. "Recent Advances in AI Planning", *AI Magazine*, 20(2), 93-123, 1999.

Wilkins, D. E. "Using the SIPE-2 Planning System", Technical Report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

Appendix A. Publications

We produced the following technical papers and reports on the project.

(1) *"A Mixed-initiative Framework for Robust Plan Sketching"*, K.L. Myers, P. A. Jarvis, W. M. Tyson, and M. J. Wolverton. In Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03), Trento, Italy, 2003.

(2) *"PASSAT: A User-centric Planning Framework"*, K. L. Myers, W. M. Tyson, M. J. Wolverton, P. A. Jarvis, T. J. Lee, M. desJardins. In Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space, 2002.

(3) *"Toward a Theory of Qualitative Reasoning about Plans"*, K. L. Myers, Technical Report, SRI International, November, 2003.

(4) "*Active Coordination of Distributed Human Planners*", K. L. Myers, P. A. Jarvis, T. J. Lee. In Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling, AAAI Press, pages 63-71, Toulouse, France, 2002.

(5) "*CODA: Coordinating Human Planners*", K. L. Myers, P. A. Jarvis, T. J. Lee, Proceedings of the European Conference on Planning, 2001.

(6) *"User's Guide for the CODA System",* K. L. Myers, P. A. Jarvis, T. J. Lee. SRI Technical Report, 2003.

(7) *"User's Guide for the PASSAT System",* K.L. Myers, P. A. Jarvis, W. M. Tyson, and M. J. Wolverton. SRI Technical Report, 2003.

Copies of (1) – (4) are included with this appendix. Documents (6) and (7) have been delivered to the government in conjunction with this Final Technical Report.

From: *Procedings of the 2003 International Conference on Automated Planning and Scheduling (ICAPS'03).*

A Mixed-initiative Framework for Robust Plan Sketching

Karen L. Myers Peter A. Jarvis W. Mabry Tyson Michael J. Wolverton

Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
{myers, jarvis, tyson, mjw}@ai.sri.com

Abstract

Sketching provides a natural and compact means for a user to outline a plan for a high-level objective. Previous work on plan sketching required that sketches be *valid*, meaning that there be at least one legal completion of the sketch relative to predefined planning knowledge. This paper addresses the problem of plan sketch interpretation when the validity assumption does not hold. We present a formal framework for robust plan sketching that defines key concepts and algorithms for interpreting and repairing plan sketches with respect to two classes of problem: violated applicability conditions and extraneous actions. We also describe a mixed-initiative implementation of this framework that supports a user and the system working collaboratively to refine a plan sketch to a satisfactory solution.

Introduction

Hierarchical planning systems support a top-down model of planning focused on the refinement of high-level objectives to executable actions. Human planners, in contrast, often combine top-down planning with a bottom-up approach that identifies specific tasks to be included in a final solution. Indeed, studies have shown that designers tend to interleave decisions at various levels of abstraction, thus working opportunistically at times rather than in a purely top-down fashion [Guindon 1990]. For example the planners of a hostage rescue may decide where and how they will establish a safe haven and how hostages will be transported, without yet having selected an overall rescue strategy. The selection of high-level strategy, in fact, can often be conditioned on such lower-level decisions.

This paper presents an HTN-based plan development framework grounded in the metaphor of *sketching*. Our approach involves having a user sketch an outline of a plan for a particular objective, with the system providing assistance in refining the outline to a full solution. A sketch consists of a collection of tasks that (1) may be only partially specified, and (2) may occur at various levels of abstraction in the plan hierarchy.[1] Within this framework, a human planner can combine opportunistic and top-down plan refinement in a manner that best suits his individual planning style. The technical challenge for sketch processing is to develop mechanisms for extending an initial sketch to a complete solution for the user's objective.

The concept of plan sketching has been considered previously [Myers 1997]. That work, however, required that plan sketches be *valid*, meaning that there be at least one legal completion of the sketch relative to predefined planning knowledge. Mismatches between human conceptualizations of a domain and formalized planning knowledge, however, can lead to situations where user sketches are uninterpretable. This paper addresses the problem of plan sketch interpretation when the validity assumption no longer holds. In particular, we present a formal framework for plan sketching that defines concepts and algorithms for interpreting and repairing invalid plan sketches in a robust manner.

Our theory of sketch interpretation and repair could be operationalized as a fully automated system. Instead, we have chosen to define a mixed-initiative approach in which the system guides a human planner through the process of modifying a plan sketch to eliminate detected problems. The role of the system in this framework is to identify sketch problems and possible repairs, while the human acts as the decision maker in navigating through the space of options.

We have implemented our robust plan sketching framework as part of a broader human-centric planning system called PASSAT *(Plan-Authoring System based on Sketches, Advice, and Templates)* [Myers et al. 2002]. Within PASSAT, users draw upon a library of *templates*, to the extent they desire, to assist with plan development. Templates are a form of task network [Tate, 1977; Erol et al. 1994], and may encode both parameterized standard operating procedures and cases corresponding to actual or notional plans developed for related tasks. PASSAT also provides a rich set of interactive and automated planning capabilities that complement the plan sketching capabilities described in this paper.

We begin the paper with a short overview of our planning model. Next, we describe the core technical components of the work, namely, a model of tolerant plan sketch compliance, a set of repair mechanisms, and a robust sketch processing algorithm. We then describe a realization of the sketch processing algorithm within

[1] Sketching often implies a graphical medium. While our model of sketching is compatible with graphical specification of tasks, we consider only logical specifications in this paper.

PASSAT's mixed-initiative planning framework, and illustrate its use in a detailed example. That is followed by a description of tools that we have built to facilitate plan sketching (namely, an interactive sketch editor and a sketch space exploration aid). Finally, we close with a discussion of related work and conclusions.

Planning Model

We employ a hierarchical task network (HTN) model of planning, based loosely on that of [Erol et al. 1994].

The cornerstones of HTN planning are *task networks* and *templates* (alternatively, operators or methods). Informally, a task network is a partially ordered set of tasks to be achieved, along with conditions on the world state before and after tasks are executed. Templates specify methods for reducing an individual task to some set of subtasks, under appropriate conditions. HTN planning consists of taking a description of an initial world state, an initial task network, and a set of templates for task refinement, and then searching for templates that can be applied to reduce the initial task network to a set of executable tasks.

Formally, we define a task network $<\mathcal{N},\mathcal{L},\mathcal{W}>$, where $\mathcal{N}$ is a set of task nodes, $\mathcal{L}$ is a set of ordering constraints on those nodes, and $\mathcal{W}$ is a set of world constraints. An HTN planning problem is defined by $<O, \mathcal{T}_0, W_0>$, where O is a set of templates, $\mathcal{T}_0$ is an initial task network, and W_0 is a set of propositions describing the initial world state. A template o is characterized by its purpose *Purpose(o)* (i.e., the tasks to which it can be applied), the preconditions for applying the template *Preconds(o)*, and the task network *Tasks(o)* to which a task matching the purpose can be reduced by applying the template. The tasks, constraints, and goals in the task networks and templates are defined using a first-order language with existential interpretation of variables.

Tasks can be either *primitive* or *nonprimitive*, with the former having no possible refinements. A solution to an HTN problem consists of a refinement of the original task network to a network of primitive tasks for which all constraints can be resolved. A solution is characterized by a *plan refinement structure* $<P,N,D>$, where P is the set of task networks produced, N is the set of nodes in any of the task networks, and D defines a directed acyclic graph of the refinement relations from a node to each of its descendants.

Each node in a plan refinement structure has attributes defined by its associated task. Key attributes for sketch interpretation include the task for the node *Task(n)*, the ancestor node *Ances(n),* the template that has been used to refine that node *Template(n)*, and the bindings for the refinement *Bindings(n)*. We use the notation p^{σ} to denote the application of bindings σ to object p (a template, term, proposition). The notation $\sigma_1 \cup \sigma_2$ denotes the composition of bindings. With this notation, $Task(n)^{Bindings(Ances(n))}$ denotes the instantiated task for node n.

Tolerant Plan Sketch Compliance

We begin by defining a plan sketch.

Definition 1 (Plan Sketch) A *plan sketch* is a set of tasks.

Note that the tasks within a plan sketch can be primitive or nonprimitive, ground or nonground.

The work in [Myers 1997] focused on the concept of *plan sketch compliance*, namely, finding a plan refinement structure that embeds an instantiation of the plan sketch. Definition 2 formalizes this requirement.

Definition 2 (Plan Sketch Compliance) A plan refinement structure $H=<P,N,D>$ is *compliant* with a plan sketch S iff there is a substitution β such that for every sketch task $A \in S$, there is some node $n \in N$ with $\sigma = Bindings\ (Ances(n))$ such that $Task(n)^{\sigma} = A^{\beta}$.

Robust plan sketching requires a less stringent condition on solutions than that of compliance from Definition 2. This weaker condition must account for both (a) user misconceptions about the task domain (i.e., situations where the user has incorrect models of when and how activities can be undertaken), and (b) background knowledge that may be incorrect or incomplete. In this paper, we focus on two types of problem within sketches that derive from user misconceptions and faulty domain knowledge:

- *Type 1:* violations of constraints from the templates used to interpret a plan sketch
- *Type 2:* sketch tasks that do not map to any high-level goal (i.e., *orphaned* tasks).

We define the weaker notion of *maximal compliance* to accommodate these problem types. In contrast to the concept of full compliance from Definition 2, maximal compliance captures the notion of embedding a maximal subset of the original sketch within a plan refinement structure while minimizing constraint violations.

The formal definition of maximal compliance builds on the concept of *conditional compliance*. Conditional compliance for a plan sketch allows a designated set of constraints to be ignored. For a set of templates O, define O/C to be the set of templates that is identical to O except that all template preconditions that unify with constraints in C have been removed.

Definition 3 (Conditional Compliance) A plan refinement structure H for a problem $<O, \mathcal{T}_0, W_0>$ is *conditionally compliant* with a sketch S and set of conditions C iff H both is compliant for S and is a plan refinement structure for the problem $<O/C, \mathcal{T}_0, W_0>$.

Definition 4 (Maximal Compliance) Let $H=<P,N,D>$ be a plan refinement structure and S_0 be a plan sketch. H is

maximally compliant with S_0 iff H is conditionally compliant with some sketch $S \subset S_0$ and conditions C, and there is no plan refinement structure H' such that for some conditions $C' \subset C$ and sketch S' where $S \subset S' \subset S_0$ either:

- H' is conditionally compliant with S' and C, or
- H' is conditionally compliant with S and C'.

Maximal compliance characterizes the class of solutions to a planning problem that best reflect a given sketch, subject to the constraints of the background knowledge. Ideally, a robust sketch interpretation algorithm should aim to identify one or more plan refinement structures that are maximally compliant. However, domain complexity may preclude finding such optimal solutions in practice.

Robust Sketch Interpretation

In this section, we define an algorithm for robust sketch interpretation that is motivated by the notions of conditional and maximal compliance. The algorithm builds substantially on the 'nonrobust' algorithm of [Myers 1997]. We first provide a high-level summary of that method, and then define a set of extensions and modifications that enable robust sketch interpretation.

Summary of the Nonrobust Method

The nonrobust method consists of two steps: (a) an initial *abduction phase* for linking sketch tasks to a high-level goal, and (b) a subsequent *refinement phase* in which the abduction results guide decision making to produce a full plan that is compliant with the sketch.

The abductive phase produces a collection of *chains* for each sketch task, where a chain encodes an abstraction path from a sketch task to a designated high-level objective through the templates defined for the planning domain.

Definition 5 [Abductive Chains] The *abductive chains* for a task A and objective G are the set of labeled linear graphs

$$\mathrm{A} = T_n \xrightarrow{[O_n,\sigma_n]} T_{n-1} \xrightarrow{[O_{n-1},\sigma_{n-1}]} T_{n-2} \xrightarrow{\Lambda} T_1 = G$$

where each O_j is a template with purpose T_{j-1} and a subtask Q_j such that σ_j is a most-general unifier of Q_j and $T_j{}^{\beta}$ for $\beta = \cup_{n \geq i > j} \sigma_i$.

We say a task A is *orphaned for an objective G* (or just *orphaned* when the objective is clear) iff there are no abductive chains linking A to G.

The abductive chains are used to guide HTN refinement in order to ensure that the resultant plan contains each of the anchors in the specified sketch. Standard task refinement involves selecting a template that applies to a given task (i.e., the template's purpose unifies with the task and all template preconditions are satisfied). For sketch processing, refinement must further restrict template choices and extend variable substitutions so that the resultant plan structure is *consistent* with at least one chain for each sketch task. Consistency requires that there be a path in the hierarchical plan structure from the top-level objective to a leaf node for which the choice of template is identical to that of the chain, and all variable substitutions are consistent. An inability to identify a compatible set of abductive chains for a refinement step indicates that the current plan cannot be expanded to a complete plan that is compliant with the original sketch; hence, further exploration of that option is pointless.

Tolerating Sketch Problems

To accommodate the two classes of sketch problem described above, we generalize and extend the nonrobust algorithm in three ways. First, violated preconditions for template application are ignored temporarily in both the abduction and refinement phases, provided they are deemed *potentially fixable* (discussed below). Second, orphaned sketch tasks are ignored during the refinement phase. Finally, a *repair phase* is added in which detected problems are resolved.

Plan Sketch Repairs

We define four types of repair: *drop constraint*, *drop task*, *modify task*, and *replace task*.

- *DropConstraint(c)* – ignore the constraint *c*.
- *DropTask(T)* – delete task *T* from the current sketch.
- *ModifyTask(T,i,v))* – change the *i*th argument of sketch task *T* to be *v*
- *ReplaceTask(T1,T2)* – replace sketch task *T1* with task *T2*

When considering repairs performed by a human (as opposed to automatically), these repair types can be categorized according to what they say about user versus system expertise. The *drop constraint* repair would be invoked in situations where the user's knowledge overrides that of the system. In contrast, application of the other repairs indicates a preference for the system's knowledge over that of the user (as reflected in his original sketch).

To provide focus, we employ two criteria to limit the applicability of repairs: (a) *relevance of the repair*, as captured by a requirement for deductive linkage between sketch tasks and violated constraints, and (b) *prespecified domain knowledge* that identifies classes of constraints and tasks to which the repairs apply.

Deductive Linkage Deductive linkage requires a logical relationship between a sketch task A and a violated constraint c through an abductive chain. Specifically, some argument to a sketch task A is connected to some argument in the violated constraint c via unification

constraints defined by the templates within the abductive chain. This linkage introduces the potential (but not a guarantee) that a change that involves the relevant sketch task argument could eliminate the violation *c*. For example, a sketch task that designates the use of a certain class of helicopter for an airlift operation might lead to violation of a constraint higher up in an abductive chain related to lift capacity. Switching to a more powerful class of helicopter could fix the problem.

In the definitions below, we use the proposition

$$Links(Task(a_1, \ldots, a_n), i, P(b_1, \ldots, b_m), Chain)$$

to indicate that within the abductive chain *Chain*, there is deductive linkage from argument a_i in $Task(a_1, \ldots, a_n)$ to b_j in some predicate $P(b_1, \ldots, b_m)$, where $P(b_1, \ldots, b_m)$ is a precondition for a template used in the chain abstraction.

Domain Knowledge Prespecified domain knowledge is used to restrict the classes of task and constraint to which various types of repair apply. We consider three categories.

A. *Droppable Constraints*. Droppable constraints correspond to predicates with a 'soft' interpretation in that they denote preferences or guidelines rather than gating conditions. For example, a template for a helicopter airlift may require wind speed below a certain threshold; a planner may decide to drop that constraint in the event that the current wind speed only slightly exceeds the threshold and all other requirements are satisfied.

B. *Modifiable Task Arguments* A task argument is categorized as modifiable to indicate that changes to that argument are allowed. For example, with the task *FLY(start, destination, flight)* in a travel planning domain, it would make sense to consider alternate flights but not start or destination locations.

C. *Replaceable Tasks* A task is categorized as replaceable to indicate that alternatives for that task can be considered. [2]

We represent these declarations as follows, using *KB* to refer to the predefined knowledge base of the planning system and x_i and y_j to denote variables. The statement

$$KB \models DroppablePredicate(P(x_1, \ldots, x_m))$$

indicates that any predicate that unifies with $P(x_1, \ldots x_m)$ is considered droppable for sketch repair; similarly

$$KB \models ChangeableTask(Task(x_1, \ldots, x_m), i)$$

indicates that the *i*th argument of any task that unifies with $Task(x_1, \ldots, x_m)$ can be modified for sketch repair, and

$$KB \models ReplaceableTask(Task(x_1, \ldots, x_m), Task(y_1, \ldots, y_n))$$

indicates that any task that unifies with $Task(x_1, \ldots, x_m)$ can be replaced by a task that unifies with $Task(y_1, \ldots, y_n)$.

We can now formally characterize the class of *induced repairs* for a given sketch and its abductive chains. The induced repairs constitute a minimal set of relevant repairs to consider when repairing a sketch.

Definition 7 (Induced Repairs) The set of *induced repairs* for a sketch *S* with abductive chains *Chains* consists of

(a) $DropConstraint(P(b_1, \ldots, b_m))$ for any unsatisfied constraint $P(b_1, \ldots b_m)$ in *Chains* such that $KB \models DroppablePredicate(P(x_1, \ldots, x_m))$

(b) $DropTask(Task(a_1, \ldots, a_n))$ for any task $Task(a_1, \ldots, a_n) \in S$ that is either orphaned, or for which there is some unsatisfied constraint $P(b_1, \ldots b_m)$ and some $C \in Chains$ such that $Links(Task(a_1, \ldots, a_n), k, P(b_1, \ldots, b_m), C)$, for some $1 \leq k \leq n$

(c) $ModifyTask(T(a_1, \ldots, a_n), i, v)$ for any task $T(a_1, \ldots, a_n) \in S$ that is orphaned, or for which $KB \models ChangeableTask(Task(x_1, \ldots, x_n), i)$ and there is some $C \in Chains$ and unsatisfied constraint $P(b_1, \ldots b_m)$ such that $Links(Task(a_1, \ldots, a_n), i, P(b_1, \ldots, b_m), C)$

(d) $ReplaceTask(Task(a_1, \ldots, a_n), Task'(b_1, \ldots, b_m))$ for any task $T(a_1, \ldots, a_n) \in S$ that is orphaned, or for which $KB \models ReplaceableTask(Task(x_1, \ldots, x_{n1}), Task'(y_1, \ldots, y_{n2}))$ and there is some unsatisfied constraint $P(b_1, \ldots, b_m)$, and $C \in Chains$ such that $Links(Task(a_1, \ldots, a_n), k, P(b_1, \ldots, b_m), Chain)$ for some $1 \leq k \leq n$

For cases (b) through (d) in Definition 7, we say that the repair *covers* the orphaned sketch task $Task(a_1, \ldots, a_n)$; for cases (a) through (d), we say that the repair *covers* the violated constraint $P(b_1, \ldots, b_m)$. The set of *potentially fixable* constraint violations is defined to be the constraint violations covered by the induced repairs.

The induced repairs provide a means to focus the repair process. Because the space of possible sketch changes can be enormous (as discussed further below), this filtering is essential for restricting the number of options considered.

Within a mixed-initiative framework, one can envision user modifications to a plan sketch that go beyond the induced repairs. Such changes could reflect additional user knowledge about the domain, or a change in strategy from that embodied in the original sketch.

[2] More generally, the properties of droppability, modifiability, and replaceability should be characterized as preference orderings. We will address this issue in future work.

ProcessSketch(S, C, <O, T_0, W_0>)

- *Step 1 [Abduction]:* Generate abductive chains *Chains(T)* for each task $T \in S$ while ignoring potentially fixable constraint violations
  - Set: *Orphans* $\leftarrow \{T \in S \mid Chains(T)=\{\}\}$
- *Step 2 [Refinement]:* Generate a task refinement structure *H* that is
  - consistent with at least one abductive chain for each $T \in S - Orphans$, and
  - ignores potentially fixable constraint violations

  If no such task refinement structure exists, then return *failure*.
  - Set: $V \leftarrow$ the potentially fixable constraint violations for *H*
- *Step 3 [Repair]:*
  - *Step 3a:* If $V=Orphans=\{\}$, then return solution <*H,S,C*>.
  - *Step 3b:* Else repair the sketch:
    - $S' \leftarrow S$
    - $C' \leftarrow C$
    - Nondeterministically select a set of induced repairs $\{r_1, \ldots r_m\}$ to cover $v \in V$ and $T \in Orphans$ If no such set exists, then return *failure*.
    - Perform the repairs as follows:
      - If $r_i = DropConstraint(v)$: $C' \leftarrow C' \cup \{v\}$
      - If $r_i = DropTask(T)$: $S' \leftarrow S' - \{T\}$
      - If $r_i = ReplaceTask(T1,T2)$: $S' \leftarrow \{S' - \{T1\}\} \cup \{T2\}$
      - If $r_i = ModifyTask(T(a_1, \ldots,a_n),i,d))$:
        $S' \leftarrow S' - \{T(a_1, \ldots,a_n)\} \cup \{T(a_1, \ldots,a_{i-1},d, a_{i+1}, \ldots,a_n)\}$
- *Step 4 [Validation]:* Invoke *ProcessSketch(S′, C′,<O/C′, T_0, W_0>)*

Figure 1. Algorithm for Robust Sketch Processing

Sketch Repair Algorithm

Figure 1 presents our algorithm for robust sketch processing.[3] Processing a sketch *S* for a problem <*O, T_0, W_0*> would involve a call to *ProcessSketch*(*S*,{},<*O, T_0, W_0*>); the results returned (via Step 3a) would be a modified sketch S^*, a set of conditions C^*, and a plan refinement structure H^* that is conditionally compliant with S^* and C^* for <*O, T_0, W_0*> (see Definition 3).

Steps 1 and 2 correspond to the abduction and refinement phases of the nonrobust algorithms from [Myers, 1997], although modified to ignore potentially fixable constraint violations and orphaned sketch tasks. Step 3 nondeterministically selects and applies induced repairs to cover all detected problems, yielding a modified sketch *S′* and collection of dropped constraints *C′*. Step 4 recursively invokes the sketch processing algorithm for *S′* and *C′* to produce a plan refinement structure that is conditionally compliant with the revised sketch (if one exists) or to identify additional problems to repair.

The algorithm as stated does not guarantee maximal plan sketch compliance (see Definition 4), although it could easily be restructured as an optimization process to identify maximal solutions. As discussed further below, we believe that optimization is an inappropriate goal because of the potentially explosive size of the repair search space. Furthermore, our experience indicates that while users prefer solutions that are close to a proposed sketch, maximal compliance is generally not necessary.

[3] To simplify the presentation, the algorithm ignores the potential for repairs that preempt each other (e.g., one repair changes an argument of a sketch task while a second replaces the sketch task with a different task).

Mixed-initiative Repair

The algorithm for sketch repair in Figure 1 does not commit to a specific implementation design. One option is to automate fully the algorithm, including the process of selecting and applying repairs. In the general case, the space of candidate sketch revisions to consider during each call to *ProcessSketch* will be of size $O(k^v)$ where *k* is the number of induced repairs for a violation and *v* is the number of violations. While *v* could be expected to be a relatively small number (say, in the range 5-10), *k* could be quite large. In particular, *modify task* repairs could encompass changes to any of a task's arguments, and may need to consider a broad range of possible values for each. A fully automated approach to sketch repair would require powerful heuristics to be effective for such a large space.

Our interests lie with more user-centric planning aids, which led us to develop a mixed-initiative realization of the sketch progressing algorithm. In our framework, the system identifies violations and possible repairs while the user selects repairs and directs the overall search. The framework is designed for iterative use, with a human planner incrementally refining a sketch in response to detected problems until finding a satisfactory solution.

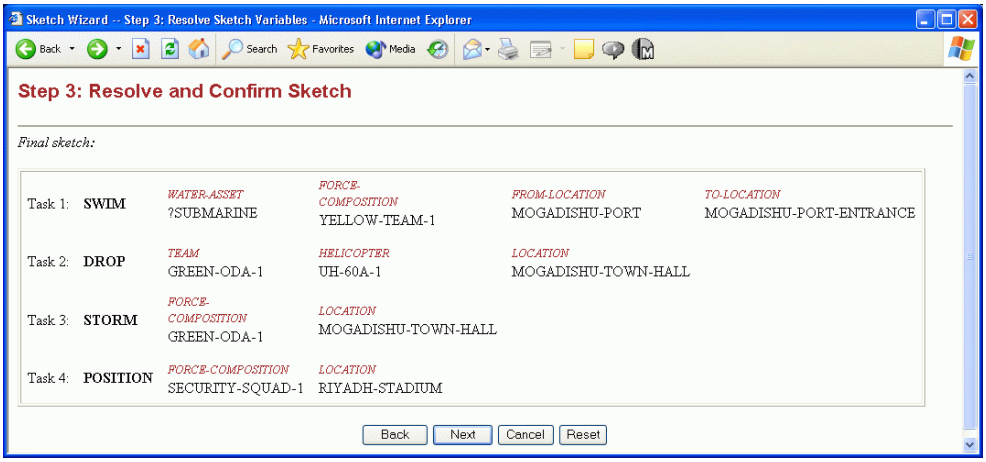

Figure 2. Sample Plan Sketch for the Hostage Rescue Task

One important characteristic of the algorithm from Figure 1 for a mixed-initiative approach is the articulation of a separate repair phase subsequent to the abduction and refinement phases. Delaying repairs until after abduction and refinement complete (as opposed to performing repairs while chains or refinement structures are constructed) means that a plan structure is available to ground the repair process. This context is important for two reasons. First, the user is not making a decision in a vacuum; rather, it is possible to understand the potential impact of a repair on the current plan. Second, interactions with the user are limited to a single candidate solution, thus providing focus. Work in the collaborative problem-solving community views focus as an essential requirement for coherent user-system interactions [Rich and Sidner, 1998]. The work of [Allen and Ferguson, 2002] similarly builds on a candidate solution (the 'straw plan') to guide mixed-initiative planning.

Our implementation differs slightly from the algorithm of Figure 1 in that it does not generate a single complete plan refinement structure in Step 2. Instead, it computes a set of *expansions*, each of which amounts to a least-commitment partial HTN structure that embeds the sketch and all derived consequences. In particular, expansions do not make commitments that are not required to connect sketch tasks to the high-level goal. For example, a sketch for a hostage rescue objective that contained only tasks related to reconnaissance would yield expansions limited to the reconnaissance subportion of the plan. This switch to expansions was motivated primarily by our desire to support a more user-centric planning process, where strategic decision making is left to the user. It would be straightforward to extend the approach to support generation of complete plans for sketches using standard HTN techniques.

Sketch Example

To illustrate the sketch-processing capabilities within PASSAT, we consider an example from a special operations domain that has motivated much of our work. The example focuses on a hostage rescue scenario in which a group of hostages is being held captive by guerrillas in Mogadishu's town hall. Riyadh Airport has been selected as the jumping-off location for the mission while the hostages are to be evacuated to Riyadh Stadium. The high-level task for this plan is represented as

```
RESCUE-HOSTAGE(MOGADISHU-TOWN-HALL,
               RIYADH-AIRPORT,
               RIYADH-STADIUM)
```

Figure 2 shows a sketch that consists of four tasks: (1) a reconnaissance force (Yellow-Team-1) swimming from a submarine (denoted by the variable `?SUBMARINE`) at Mogadishu Port to the port entrance, (2) inserting a combat team (Green-ODA-1) at the town hall via a UH-60A helicopter, (3) having the combat team storm the town hall, and (4) positioning a security team at the evacuation site. The labels above each task argument identify that argument's role in the task.

Processing of this sketch by PASSAT yields three expansions, with a range of three to four violated constraints in each. The expansions interpret the role of the sketch tasks somewhat differently; for example, one expansion interprets the `DROP` task as part of the hostage extraction effort while the others interpret it as part of a reconnaissance operation.

The user can select one of these expansions and explore options for repairing its associated problems. Figure 3 summarizes the constraint violations (top) and the hierarchical task/template structure (bottom) for one expansion; sketch tasks are highlighted. This expansion does not contain any orphaned tasks.

Figure 4 displays the window that is presented to the user to repair the original sketch. The window summarizes the repair options for each violation, which may consist of dropping the constraint, changing a parameter for a designated task, or making no repair. (Our interface does not yet support *replace task* repairs.) Because the use of

Violated Constraints

VC1. (SITUATION-TYPE RIYADH-STADIUM HOSTILE)
VC2. (DISTANCE-< RIYADH-AIRPORT MOGADISHU-TOWN-HALL (RANGE UH-60A-1)
VC3. (> (SEA-TEMPERATURE MOGADISHU-PORT-ENTRANCE) 40)
VC4. (PLATOON-SIZED SECURITY-SQUAD-1)

Expansion Task Structure

Task: RESCUE-HOSTAGE(MOGADISHU-TOWN-HALL, RIYADH-AIRPORT, RIYADH-STADIUM)
Template: Hostage-Recovery-To-A-Potentially-Unstable-Area
  Task: ADVANCED-RECON(COUNTRY-OF(MOGADISHU-TOWN-HALL))
  Template: Advanced-Recon-Of-Target-Area
    Task: RECON-SEAPORTS(SOMALIA)
    Template: Recon-Seaports-In-Area
      Task: RECON(MOGADISHU-PORT)
      Template: Recon-With-Covert-Ground-Force
        Task: EXFILTRATE(YELLOW-TEAM-1, MOGADISHU-PORT, *?TO-LOC*)
        Template: Swim-Exfiltrate-To-Submarine
          Task: SWIM(*?SUBMARINE*, YELLOW-TEAM-1, MOGADISHU-PORT, MOGADISHU-PORT-ENTRANCE)
  Task: RESCUE-AND-RECOVER(*?FORWARD-POINT*, MOGADISHU-TOWN-HALL, *?RECOVERY-LOCATION*)
  Template: Rescue-And-Recover-Hostages
    Task: STORM(GREEN-ODA-1, MOGADISHU-TOWN-HALL)
    Task: INFILTRATE(GREEN-ODA-1, RIYADH-AIRPORT, MOGADISHU-TOWN-HALL)
    Template: Helicopter-Insertion-Rope
      Task: DROP(GREEN-ODA-1, UH-60A-1, MOGADISHU-TOWN-HALL)
  Task: PROVIDE-SECURITY(RIYADH-STADIUM)
  Template: Site-Defense-Large-Reaction-Force
    Task: POSITION(SECURITY-SQUAD-1, RIYADH-STADIUM)

Figure 3. Violated Constraints and Plan Structure for the Selected Expansion

constraint dropping and task parameter changes is restricted by predefined domain knowledge, only some of these repairs may apply in each case.

To support the user in changing a task parameter, the interface provides a drop-down list of candidate values. This set consists of instances for the type associated with that argument, with values that lead to violation of the given constraint (in accord with the deductive linkage from the sketch task to the constraint) explicitly marked as such.

As one approach to repairing the chosen expansion, the user could perform the following repairs:

- drop the constraint *VC1*
- modify the *Helicopter* argument of the DROP task to be UH-60L-1 rather than UH-60A-1, given that UH-60Ls have greater range (to address the violated constraint *VC2*)
- drop the constraint *VC3*
- modify the *Force-Composition* argument of the POSITION task to be SECURITY-PLATOON-1 (to address the violated constraint *VC4*)

Given a set of repairs, PASSAT attempts to validate the revised sketch by reinterpreting it while ignoring the dropped constraints. In this case, the repairs resolve the original problems but introduce a violation of the constraint (COMBAT-EFFECTIVE SECURITY-PLATOON-1). This new problem can be repaired by changing the *Force-Composition* argument to be SECURITY-PLATOON-2 (i.e., a platoon that has been certified ready for combat). Processing of this revised sketch yields a single expansion with no constraint violations.

Figure 5 displays a snapshot of PASSAT's interface after sketch processing has completed. The large frame on the left contains a hierarchical decomposition of the current plan refinement structure, showing the insertion of the final expansion for the Hostage-Rescue task. Items next to folder icons are tasks that have been expanded; items next to star icons are nonprimitive tasks that can be expanded further; items next to document icons are primitive tasks. Sketch tasks appear italicized and highlighted in bold font. The frame on the upper right shows the current *agenda* – the list of planning steps the user must perform to address outstanding issues. PASSAT maintains this agenda automatically to assist a user in managing the planning process. Constraints that the user chose to drop as part of the repair process appear highlighted on the agenda. Planning tasks that remain to be expanded are also added to the agenda. The frame on the lower right shows the list of *information requirements* – sources of information that have been identified by the user or PASSAT's planning knowledge as relevant to the planning process.

At this stage, the user could continue developing the current plan, by using any of PASSAT's capabilities for interactive planning, or by providing a plan sketch for a nonprimitive task.

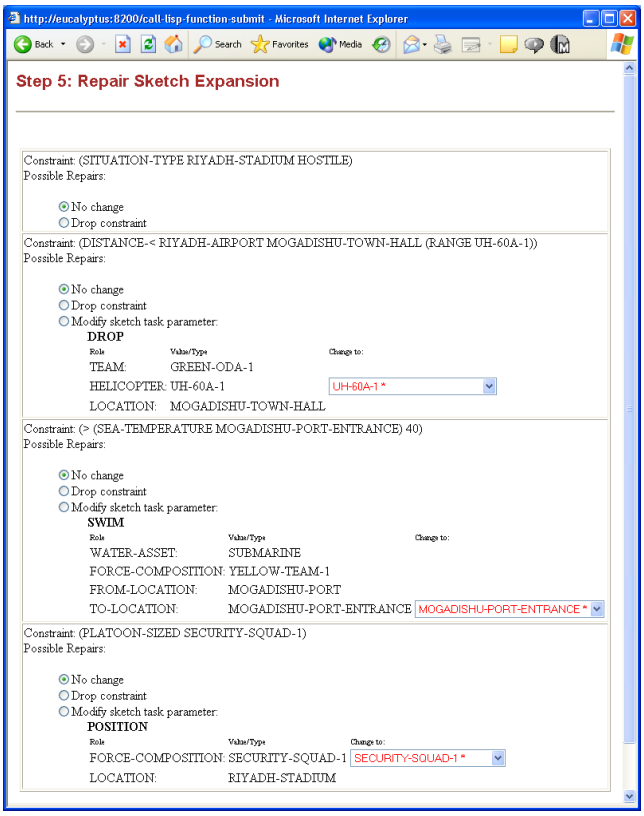

Figure 4. Candidate Repair Options

Sketching Tools

Mixed-initiative systems require powerful and flexible interfaces to facilitate interactions with a user. To support mixed-initiative sketch repair, we developed two interactive tools: a *sketch editor* and a *sketch space exploration aid*.

Sketch Editor

Sketch specification involves defining the tasks that comprise a sketch and their arguments. PASSAT provides an interactive editor to simplify this process. With this editor, the user first selects a set of tasks to be included in the sketch, and then specifies the arguments for those tasks. Allowed arguments consist of variables and all instances of the corresponding type for that argument. Figure 3 displays a final sketch created within the editor.

To help the user focus on relevant choices, the sketch editor incorporates context-sensitive presentation of options to the user for both task and argument selection.

- *Task selection:* The editor exploits linkage among templates to limit task selection for a sketch to tasks that could possibly appear in any expansion of the 'objective' currently under consideration. This filtering helps to eliminate many irrelevant options, thus both reducing clutter from the task selection menu and preventing the user from pursuing many fruitless avenues.

- *Argument selection:* It is often the case that many

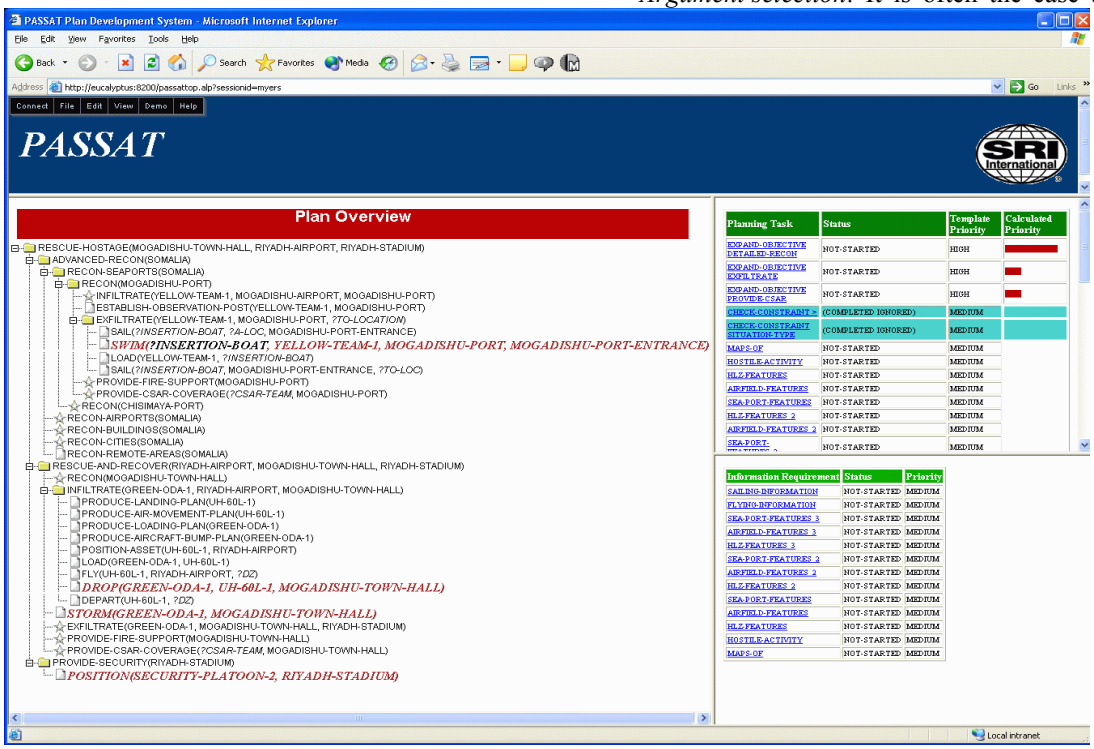

Figure 5. Plan with Sketch Expansion

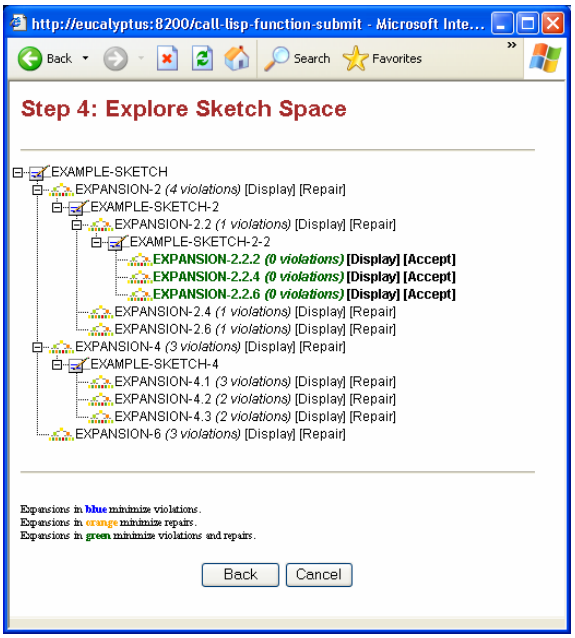

Figure 6. Sketch Space Exploration Tool

candidate values for a task argument fail to satisfy the preconditions of any templates that could be applied to the task. Eliminating such values from consideration prevents exploration of many dead-ends. However, one design requirement for PASSAT was the flexibility to let a user think 'out of the box'. In particular, PASSAT's constraint reasoning allows certain constraints to be overridden at the user's discretion. For this reason, the possible values presented to the user are flagged to indicate whether or not they satisfy all associated constraints.

This type of structured plan editor eliminates the possibility of syntactic mistakes (e.g., undefined tasks or arguments, use of inappropriate argument types) that can be a source of great frustration to a user. In doing so, it allows the user to focus on the conceptual design for a sketch.

Sketch Space Exploration Tool

The space of possible expansions for a given sketch can be dauntingly large, especially when interpretation is tolerant of invalid sketches. To support a user in navigating this large space, we have developed a *sketch space exploration* tool that aids a user in managing the sketch refinement process (see Figure 6). The tool is organized around a tree structure that reflects the space of sketches and expansions that a user has explored. The root of the tree corresponds to the initial sketch; it contains a descendant node for each expansion of the sketch. Each revision of an expansion in turn generates a descendant sketch node, from which a recursive structure emerges.

For a sketch node, the user can choose to generate expansions all at once or incrementally. For an expansion, a user can view the template structure and the detected problems. Expansions with minimal problems and minimal numbers of expected repairs to address those problems are highlighted. (One repair could fix multiple problems, thus these values can differ for a given expansion.) Eventually, the exploration tool will contain mechanisms to summarize and compare expansions and sketches.

Related Work

The NuSketch system [Forbus eta al. 2001] provides a framework for creating graphical sketches of plans (specifically, for military courses of action) via a drawing metaphor. As with our work, these sketches are intended to provide outlines rather than complete plans, but in a pictorial rather than logical language. NuSketch is focused on interpretation of visual inputs and the adequacy of mechanisms for specifying sketches visually, in contrast to our emphasis on interpreting sketches relative to a knowledge base of plan templates and helping a user refine a sketch to a satisfactory solution.

Qu and Beale's work on cooperative response generation provides a mixed-initiative framework for constraint-based variable assignment problems [Qu and Beale 1999]. Users can perform 'repairs' by changing selections or dropping constraints. The system detects violations and can assist the user by proposing new values and summarizing possible solutions. While similar to our mixed-initiative sketch repair, this work does not incorporate any notion of plans. The authors note that, while there has been much work on cooperative response generation, most of it does not consider interactions among choices.

Our work on sketch interpretation shares with plan recognition techniques the objective of finding a plan that 'covers' a set of specified tasks (see [Carberry, 2001] for a comprehensive overview of the field of plan recognition). These two lines of work differ, however, in several respects. One difference is that the plan recognition work is grounded in the assumption that there is a single intended plan to be determined; in contrast, our work supports the more general notion of identifying a range of possible interpretations for a given sketch. A second fundamental difference relates to the starting point: plan recognition techniques assume a complete, ordered set of tasks for a plan, while our model of a plan sketch consists of a partial and unordered set of tasks. In particular, plan recognition work does not consider the problem of extending a partial plan to a complete solution. Finally, most plan recognition work has been done in the context of STRIPS models of planning, in contrast to our focus on HTN models (although see [Gertner and Webber, 1996] for another HTN-based approach).

Most plan recognition work has assumed that observed actions (the analog of our sketch tasks) are part of a valid

plan for an undetermined goal. However, there have been some notable attempts to address the problem of recognition of faulty plans. Classifications of different types of plan-based misconceptions are presented in [Pollack 1986, Quilici et al. 1988, van Beek et al. 1993], with a comprehensive and detailed list provided in [Calistri-Yeh 1991]. The emphasis in that work is on identifying user misconceptions, with no consideration given to potential problems in the underlying domain knowledge. Misconceptions can be broadly characterized in terms of missing actions, violated/unsupported preconditions, and unsupported actions. Within the context of plan sketching, only violated preconditions and unsupported actions make sense (since the plan is only partially specified). Many of these papers also present methods to detect misconceptions and (at times) suggest potential fixes. [Calistri-Yeh 1991] incorporates a probabilistic model of a user to identify 'more likely' explanations for observed actions. Such a model could be useful within the context of our work to focus the user on expansions and repairs with greater expected relevance.

In the long term, we are interested in tools that support user updates to background planning knowledge when gaps or errors are detected. [Cohen and Spencer 1994] present an ATMS-based method for incremental updates to plan recognition structures when knowledge is added.

Conclusions

Plan sketching provides a powerful paradigm for user specification of complex plans. Plan sketching can help a user quickly outline the key aspects of the plan, capitalizing on the system to fill in less important details around the sketch. In addition, it can serve as the basis for an exploratory process that allows a user to consider a variety of options when developing a plan.

Robustness is critical to ensuring that a plan sketching tool is usable and helpful. Robustness requires the ability to identify differences between a user's outline for a plan and what the planning knowledge within the system supports as possible, as well as mechanisms to address those problems.

The work presented here has defined an approach to robust plan sketch interpretation that accommodates two categories of problem: violated applicability conditions and extraneous actions. This approach has been embodied within a mixed-initiative plan sketching framework in which a system identifies options for repair while a user selects candidate interpretations and repairs.

Areas for further work include broadening a sketch to include user constraints and temporal information, and developing tools to improve user understanding of the sketch space (specifically, summarization and comparison tools for sketches and expansions).

Acknowledgments. This work was supported by DARPA under Air Force Research Laboratory Contract F30602-00-C-0058.

References

Allen, J. and Ferguson, G. (2002). Human-machine Collaborative Planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Calistri-Yeh, R. (1991). Utilizing User Models to Handle Ambiguity and Misconceptions in Robust Plan Recognition. *User Modeling and User-Adapted Interaction*, 1(4).

Carberry, S. (2001). Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11(1-2).

Cohen, R. and Spencer, B. (1994). Specifying and Updating Plan Libraries for Plan Recognition Tasks. In *Proceedings of IEEE Conference on Tools for AI*.

Erol, K., Hendler, J., and Nau, D. (1994). Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Gertner, A. S. and Webber, B. L. (1996) A Bias towards Relevance: Recognizing Plans where Goal Minimization Fails. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Guindon, R. (1990). Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 5(2).

Forbus, K. D., Ferguson, R. W., and Usher, J. M. (2001). Towards a Computational Model of Sketching. In *Proceedings of Intelligent User Interfaces*, Sante Fe, New Mexico.

Myers, K. L. (1997). Abductive Completion of Plan Sketches. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, AAAI Press.

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., and desJardins, M. (2002). PASSAT: A User-centric Planning Framework. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Pollack, M. (1986). A Model of Plan Inference that Distinguishes between the Beliefs of Actors and Observer. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguisitics*, N.Y., N.Y.

Qu, Y. and Beale, S. (1999). A Constraint-Based Model for Cooperative Response Generation in Information Dialogues. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI Press.

Quilici, A., Dyer, M. G., and Flowers, M. (1988). Recognizing and Responding to Plan-Oriented Misconceptions. *Computational Linguistics*, 14(3).

Rich, C. and Sidner, C. L. (1998). COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*, 8(3-4).

Tate, A. (1977). Generating Project Networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*.

van Beek, P., Cohen, R., and Schmidt, K. (1993). From Plan Critiquing to Clarification Dialogue for Cooperative Response Generation. *Computational Intelligence*, 9(2).

From: *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.*

PASSAT: A User-centric Planning Framework

Karen L. Myers[1] W. Mabry Tyson[1] Michael J. Wolverton[1]
Peter A. Jarvis[1] Thomas J. Lee[1] Marie desJardins[2]

[1] Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
{myers,tyson,mjw,jarvis,tomlee}@ai.sri.com

[2] University of Maryland, Baltimore County
Dept. of CS and EE
1000 Hilltop Circle
Baltimore, MD 21250
mariedj@cs.umbc.edu

Abstract

We describe a plan-authoring system called PASSAT *(Plan-Authoring System based on Sketches, Advice, and Templates)* that combines interactive tools for constructing plans with a suite of automated and mixed-initiative capabilities designed to complement human planning skills. PASSAT is organized around a library of predefined *templates* that encode task networks describing standard operating procedures and previous cases. Users can select from these templates to apply during plan development, with the system providing various forms of automated assistance. A mixed-initiative *plan sketch* facility helps users refine outlines for plans to complete solutions, by detecting problems and proposing possible fixes. An *advice* capability enables user specification of high-level guidelines for plans that the system helps to enforce. Finally, PASSAT includes *process facilitation* mechanisms designed to help a user track and manage outstanding planning tasks and information requirements, as a means of improving the efficiency and effectiveness of the planning process. PASSAT is designed for applications for which a core of planning knowledge can be captured in predefined action models but where significant user control of the planning process is required.

Introduction

AI planning technology provides powerful tools for solving problems that require the coordination of actions in the pursuit of specified goals. To date, however, there has been limited success in transitioning this technology to significant applications in the commercial, military, or space sectors. A major obstacle to technology transfer lies with the lack of control available to potential users of planning systems. AI planning systems have traditionally been designed to operate as *black boxes*: they take a description of a domain and a set of goals and automatically synthesize a plan for achieving the goals. Human planners, however, are generally reluctant to cede full control to automated planning systems in this manner.

Many potential consumers of planning technology require more *user-centric* tools that are designed to augment human skills rather than replace them. This observation has led, in recent years, to the development of a number of *plan-authoring* frameworks. Plan-authoring systems provide a set of plan editing and manipulation capabilities that support users in developing plans. These systems introduce a degree of structure to the planning process, yielding principled representations of plans with well-defined semantics. Plan-authoring systems can include a range of planning aids that reason over this structure; however, the role of such automated aids is to augment human planning skills by facilitating human-driven plan development. Interest in plan-authoring systems is strong within both the space and military sectors, for their potential to improve the quality and process of plan development without incurring the high knowledge modeling costs and loss of control associated with fully automated planning systems.

This paper describes a plan-authoring system called *PASSAT (Plan-Authoring System based on Sketches, Advice, and Templates)* designed to support user-centric planning. At its heart, PASSAT is a plan-authoring system in which users construct and modify plans interactively. Users can draw upon a library of *templates*, to the extent they desire, to assist with plan development. Templates correspond to a form of hierarchical task network (HTN) [Tate, 1977], and may encode both parameterized standard operating procedures and cases corresponding to actual or notional plans developed for related tasks.

To complement these interactive tools, PASSAT includes a range of automated and mixed-initiative planning capabilities. Users can invoke an automated planning mode based on standard HTN methods to expand any open task within a plan. A *plan sketch* facility enables users to create outlines of plans that are then filled out using templates designed for similar tasks. *Advice* within PASSAT enables users to define high-level policies to be satisfied by both plans and planning processes. Such guidance can be useful both in directing automated components within the system, and in tracking high-level guidelines that a user wants satisfied but may inadvertently violate through his interactive planning choices.

PASSAT also includes *process facilitation* mechanisms designed to aid the user in managing plan development. These mechanisms help the user track open tasks and

outstanding information requirements for the current plan. Such assistance is critical in complex applications, as it helps the user stay focused without overlooking important details.

With its combination of interactive and automated capabilities, PASSAT enables a user to quickly develop plans that draw upon past experience encoded in templates but are customized to his individual preferences and the demands of the current situation. PASSAT has been under development for about a year. This paper describes both the current PASSAT system and the more comprehensive plan-authoring system toward which we are working (with all future work noted explicitly as such). We begin with a more detailed discussion of PASSAT and an example of its use, followed by a description of the representational constructs within PASSAT, the user-centric planning capabilities, and the process facilitation mechanisms. The final section discusses related work.

PASSAT Overview

Plan development within PASSAT has been guided by two key principles:

- *Flexible, 'out of the box' planning:* Traditional AI planning systems lock users into a set of solutions, namely, those implied by the predefined action models that underlie plan development. Within PASSAT, templates are viewed as guidelines for performing tasks; the human planner is free to expand the set of solutions defined by the templates. In particular, a user can override constraints, drop tasks, or insert additional tasks in accord with his personal preferences or the demands of the current situation. Such flexibility is critical for domains in which correct and comprehensive collections of templates cannot be provided.
- *Controllable user-centric automation:* Automated capabilities within PASSAT are designed both to complement human planning skills and to be readily directable by a human. Automation would be invoked under user control only in contexts where he feels that it would be beneficial.

Domain Characteristics

PASSAT is generic, domain-independent technology but is tailored toward applications with the following characteristics.

(a) The complexity of the domain precludes full capture of all relevant planning knowledge. However, partial planning models can be developed.

(b) Human input is critical, but some amount of automation would both improve plan quality and reduce overall planning time.

Our motivating application domain, Special Operations Forces (SOF) mission planning, has these characteristics. Standard operating procedures exist for many high- and mid-level activities in the SOF domain, and are readily amenable to encoding within an HTN representation. For example, a hostage rescue operation can be characterized as consisting of the high-level objectives of performing reconnaissance in the areas around the rescue site, establishing a safe haven to which to remove the hostages, undertaking the assault to rescue the hostages, and transporting the hostages to the safe haven. Low-level operations follow standard doctrine and can also be modeled in a relatively straightforward manner.[1] Intermediate strategy decisions pose a bigger challenge. For example, informed selection of areas and methods for reconnaissance requires deep background knowledge of reconnaissance operations, breadth of understanding of the current situation, and significant experience. Capturing and modeling this type of strategic knowledge in full presents a tremendous challenge.

SOF planning lies well beyond the range of current automated planning technologies; moreover, fully automated solutions are unlikely ever to succeed because of the difficulty in formulating strategic knowledge with sufficient fidelity. In contrast, a PASSAT-style plan-authoring system provides a good technological match for the SOF planning domain. Missions arise unexpectedly, resulting in a need to assemble high-quality plans rapidly. Thus, the availability of tools to expedite plan development is important. Because many types of SOF operations can be broadly characterized with predefined templates, knowledge bases can be developed that capture certain portions of the planning process. However, individual operations tend to be highly distinctive, making it important to have tools that enable users to modify and customize plans to suit the needs of a particular situation.

Many potential application domains for planning technology share these characteristics of having partially formalizable domain knowledge and requiring significant user input to produce high-quality, situation-specific plans. On the military side, examples include air operations, disaster relief planning, and noncombatant evacuation operations. Space applications include science mission planning and ground operations planning.

PASSAT Example

Figure 1 shows a snapshot of the PASSAT interface during a planning session. The large frame on the left contains a hierarchical decomposition of the current partial plan. Items next to folder icons are tasks that have been expanded; items next to star icons are tasks that can be expanded further (either through automated template application or interactively); and items next to document icons are tasks that match no templates. The frame on the

[1] Many of our templates were derived directly from SOF field manuals.

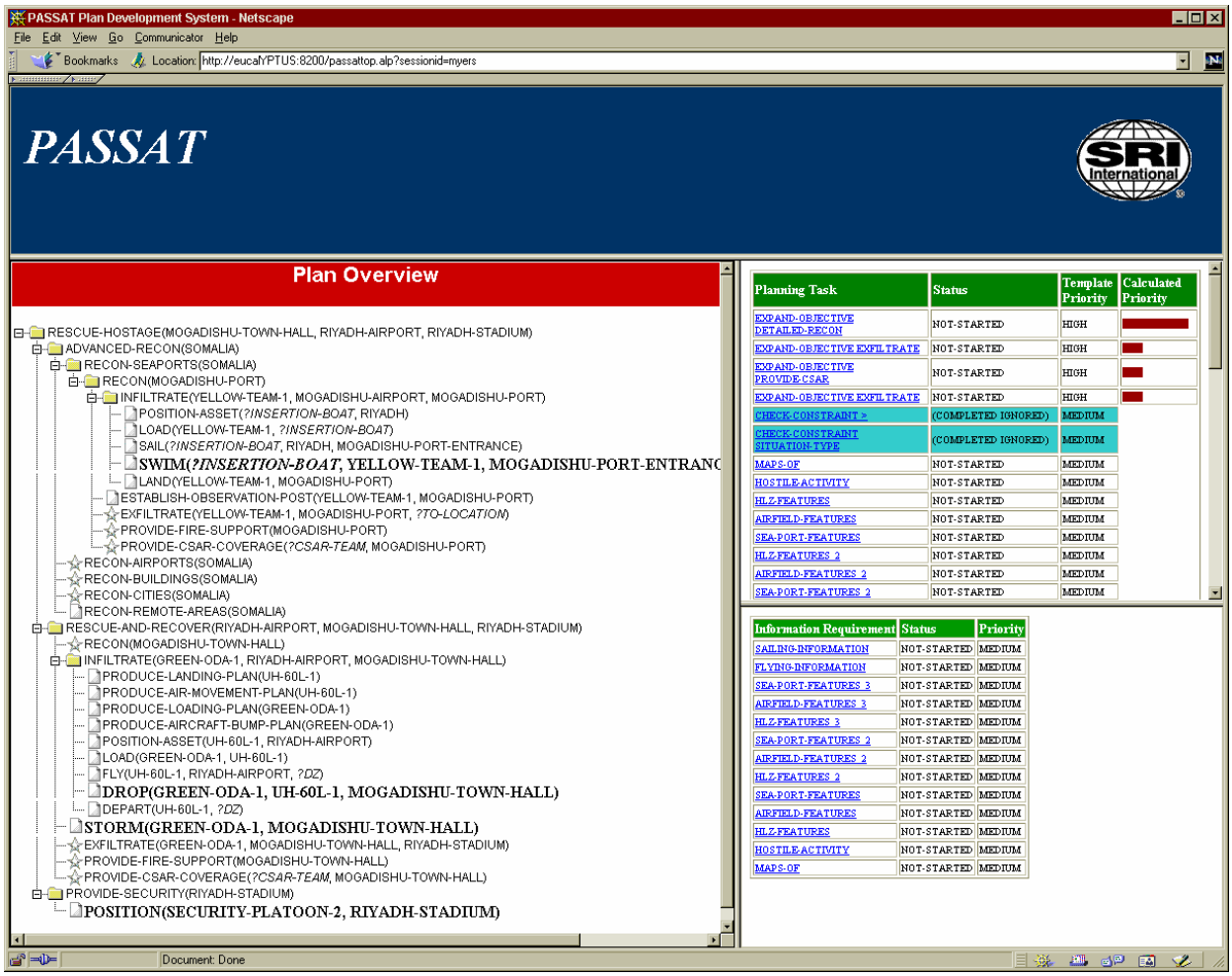

Figure 1. PASSAT Interface during Plan Development

upper right shows the current *agenda* – the list of planning steps the user must perform to address outstanding issues. The frame on the lower right shows the list of *information requirements* – sources of information that have been identified by the user or PASSAT's planning knowledge as relevant to various portions of the planning process.

The human planner develops the plan by selecting a planning step from the agenda and performing that step (many of these planning steps are accessible through the plan display as well). If the planning step is to expand the `PROVIDE-CSAR-COVERAGE` task, for example, the planner would be presented with several options: apply one of the templates that matches the task (see Figure 2), enter an expansion manually, or create a sketch for achieving the `PROVIDE-CSAR-COVERAGE` task and work with PASSAT to refine that sketch. Performing this planning step may cause additional planning steps to be added to the agenda (i.e., new tasks, variables, and constraints may have been introduced into the plan) and new information requirements as well.

Plan Representation

PASSAT's representation of plans and tasks is based on a fairly standard HTN model (similar to that of [Erol et al., 1994]), augmented with a rich temporal representation for tasks. Using PASSAT, a user would describe the objective of the plan in the form of one or more *task statements*, each consisting of a *task operator* and *terms* (variables, instances, or functions applied to terms).

Templates A *template* describes one way that a task (i.e., the template's *purpose*) can be decomposed into subtasks. A template consists of a set of these *subtasks*, as well the *variables* used in the template, *constraints* on the applicability of the template, and the *effects* of successfully performing individual tasks and the entire template. Different templates may describe different decompositions for the same task.

Figure 2. A Candidate Template for Task Refinement

PASSAT's template representation supports two features not found in the framework of [Erol et al. 1994], namely *information requirements* (discussed in detail below) and *enumeration* tasks. Enumeration tasks enable the specification of a set of tasks relative to a set of terms that satisfy a designed predicate. For example, the enumeration task

$$\forall ?city.\mathrm{DISTANCE}(?city, ?hostage\text{-}locn) < 20 \Rightarrow \mathrm{RECON}(?city)$$

indicates that a RECON task should be performed for each city within the specified distance. Other HTN frameworks (e.g., O-Plan [Currie and Tate, 1991] and SIPE-2 [Wilkins 1993]) provide similar mechanisms for enumerating subtasks relative to a designated constraint.

Constraints Constraints consist of state predicates that denote hard or soft conditions, perhaps due to physical laws or policy rules. PASSAT employs a three-valued logic for constraints, grounded in the values TRUE, FALSE, and UNKNOWN.

Automated constraint checking is performed when constraints are created or modified in the plan. Checking of ground constraints may return a status of UNKNOWN, if the information is not specified in the world state; such constraints would need to be validated explicitly by the user. Checking of nonground constraints occurs only when the number of possible instantiations is less than a predefined threshold, with the system testing whether the constraint is valid or invalid for each (i.e., establishing that the constraint is necessarily true or false independent of the instantiation). Otherwise, the system returns UNKNOWN and the constraint is rechecked when more variables are instantiated.

Unlike in automated planning systems, a constraint with value other than TRUE does not necessarily halt the process or cause backtracking. Instead, a violated constraint is called to the attention of the user, who has the choice of ignoring the violation or changing the step that triggered the violation.

Temporal Representation PASSAT supports the scheduling of tasks via constraints on the earliest and latest possible times for the start and end points of tasks. Temporal constraints typically refer to these end points but may also refer to upper and lower bounds on those time points. Temporal constraints can also be expressed using Allen's interval relations [(Allen, 1984)].[1]

Domain Definition PASSAT utilizes a number of coordinated databases to define its application domain. An *ontology* (based on the Generic Frame Protocol representation [Karp et al., 1995]) defines the hierarchical organization of classes and instances and their properties. *State predicate* and *task statements* are declared, specifying the number and classes of their arguments. *Functions* are similarly declared, with the additional declaration of the class of the function's value. Some predicates and functions are computable (e.g., <, +, and Distance) while others are defined by their extent. The *world state* is defined by a set of ground state predicates.

User-centric Plan Development

PASSAT currently provides two main modes of plan development: *interactive plan refinement* and *plan sketching*. Future versions of PASSAT will also support an advice module to guide plan development.

[1] The temporal reasoning portion of the system is not yet fully implemented.

Interactive Plan Refinement

Interactive plan refinement in PASSAT involves three types of planning step: *expand task*, *instantiate variable*, and *resolve constraint*.

Expand Task When a task is to be expanded, the system offers the user the choice of applying a predefined template, specifying a set of subtasks interactively, sketching a solution (see below), or dropping the task.

When the user chooses a template to apply, the system first unifies the task and the template's purpose, making appropriate substitutions throughout the template. PASSAT adds the (partially instantiated) subtasks and constraints of the template to the plan. In addition, it extends the agenda to include planning steps to expand the new subtasks, to check the new constraints, and to instantiate any unbound variables from the template. The planning step for the parent task is marked as completed and removed from the agenda. In the displayed plan, the parent task is shown with its subtasks.

As the system performs this step, it also checks the status of all new constraints. If one is found to be valid, the planning step to check it is marked as completed and removed from the agenda. If it is found to be invalid, the planning step is flagged.

As the system expands a task, other planning steps may be affected. If the unification results in the assignment of a value to a variable, the planning step for instantiating that variable is removed. The status of constraints that contained that variable might now be resolvable; the system checks those constraints and updates the planning steps, if necessary.

Instantiate Variable The agenda contains a planning step for each unbound variable within the current plan. When the user is ready to instantiate a variable, PASSAT provides the set of possible instantiations that satisfy all relevant constraints; the user can select from this set, provide an alternative value (hence, overriding a relevant constraint), or simply mark some subset of the values as unacceptable. When the variable is instantiated, any impacted constraints are rechecked. A user can optionally provide a justification (currently, a text string) for his actions.

Resolve Constraint As noted above, PASSAT provides automated checking of constraints as part of template application, with the agenda being used to track constraints that the system was unable to validate. *Resolve constraint* steps enable a user to declare that the system can disregard individual constraints with the status of `FALSE` or `UKNOWN` in a given situation. Such declarations do not have assertional import (i.e., they do not change the system's world model); rather, they enable relaxation of constraints from the planning model embodied in the domain templates. A user can declare that a given constraint be ignored for a variety of reasons: (a) he has more recent information that would validate the constraint, (b) he knows that the constraint is overly strong for the current situation, or (c) he wants to explore a *what-if* scenario. PASSAT supports the user in providing a justification (currently, a text string) for such constraint relaxations.

Robust Plan Sketching

Hierarchical planning systems are designed to support top-down development of plans, taking an initial high-level objective and refining it to increasingly more concrete levels. Human planners, in contrast, often combine refinement-style planning with a more bottom-up approach that identifies specific tasks to be included in a final solution. For example, the planners of a hostage rescue may know where and how they will establish a safe haven without yet having decided on a particular high-level rescue strategy.

Within PASSAT, a user can *sketch* an outline of a plan, with the system providing assistance in expanding the sketch to a full-fledged solution for a particular objective. A sketch consists of a collection of tasks that (1) may be only partially specified, and (2) may occur at various levels of abstraction in the plan hierarchy. When given a sketch, PASSAT generates possible *sketch expansions*, which correspond to least-commitment plan structures that embed the sketch and all derived consequences. The user may choose any of these expansions to continue planning; the agenda will be updated to reflect the derived set of outstanding tasks.

The sketch processing capability within PASSAT builds substantially on the algorithms of [Myers, 1997] but provides robustness through an ability to recognize and respond to *invalid* sketches. By invalid, we mean a sketch for which there is no legal completion relative to the set of defined templates. To provide robustness in the face of invalid sketches, the sketch completion algorithm has been extended to tolerate constraint violations that are classified as *potentially fixable* according to prespecified domain knowledge about constraints and tasks (discussed further below). PASSAT guides the human planner through the process of repairing fixable constraint violations within expansions that he selects. Users can select from two types of repair method: *constraint drop* and *task modification*.

Constraint drop repair involves simply ignoring the violated constraint; this type of repair is appropriate for constraints with a 'soft' interpretation (i.e., they correspond to preferences or guidelines rather than gating conditions). For example, a template for a helicopter airlift may require wind speed below a certain threshold; a planner may decide to drop that constraint in the event that the current wind speed only slightly exceeds the threshold and all other requirements are satisfied. Constraint drop repair can be applied only to constraints that have been explicitly declared as ignorable for the sake of sketch repairs.

Task modification involves changing one or more arguments of a sketch task that are deductively linked to a

Figure 3. Sample Plan Sketch for the Hostage Rescue Task

violated constraint. For instance, consider a sketch that contains two tasks: the establishment of a safe haven at a particular location, and a helicopter airlift to remove rescued hostages to the safe haven. If the helicopter has insufficient range to reach the safe haven, the user would be given the options of selecting an air asset with appropriate range characteristics, or choosing a closer destination for the safe haven.[1] Domain knowledge restricts the set of arguments that can be modified in service of sketch repair, as a means of limiting the number of options to consider (both by the user and the system).

The robust plan sketch capability within PASSAT is designed to be used iteratively, with a human planner repeatedly refining a sketch in response to detected problems until a solution is found that meets his needs.

Sketch Example To illustrate the sketch-processing capabilities within PASSAT, we consider an example from a hostage rescue scenario in which a group of Americans is being held captive by guerrillas in Mogadishu's town hall. Riyadh Airport has been selected as the jumping-off location for the mission while the hostages are to be evacuated to Riyadh Stadium. The high-level task for this plan is represented as

```
RESCUE-HOSTAGE(MOGADISHU-TOWN-HALL,
               RIYADH-AIRPORT,
               RIYADH-STADIUM)
```

PASSAT provides an interactive editor for specifying a plan sketch. Figure 3 shows a completed sketch consisting of four tasks: (1) having an infiltration team (Yellow-Team-1) swim from a submarine (denoted by the variable `?SUBMARINE`) located at the entrance to Mogadishu Port to the port itself, (2) inserting a combat team (Green-ODA-1) at the Town Hall via a UH-60A helicopter, (3) having the combat team storm the Town Hall, and (4) positioning a security team at the evacuation site. The labels above each task argument identify that argument's 'role' in the task.

Processing of this sketch by PASSAT yields six expansions, with a range of three to four violated constraints in each. The user can select one of these expansions and explore options for repairing its associated constraint violations. Figure 4 summarizes the constraint violations and the hierarchical template structure for one of the expansions.

Figure 5 displays the window that would be presented to a user to assist in the repair of the original sketch. The window summarizes the available repair options for each violation, which may consist of dropping the constraint, changing a parameter for a designated task, or making no repair. Because the use of constraint dropping and task parameter changes is restricted (by predefined domain knowledge about their applicability), these repairs are not necessarily applicable in each case.

To support the user in changing a task parameter, the interface provides a drop-down list of candidate values. This set consists of instances for the type associated with that argument, filtered to remove values that lead to violations of the given constraint (in accord with the deductive linkage from the sketch task to the constraint). This filtering is incomplete: the list may include values that do not fix the detected problem, due to interactions with constraints in other parts of the plan. Future versions of PASSAT will incorporate additional checking to restrict this set further.

To repair the chosen expansion, the user could perform the following repairs:

[1] A sketch could also be repaired by changing the type of a task, rather than simply changing the task arguments (e.g., ground-based evacuation rather than an airlift). PASSAT does not currently support this class of sketch repair.

Violated Constraints

VC1. `(SITUATION-TYPE RIYADH-STADIUM HOSTILE)`
VC2. `(DISTANCE-< RIYADH-AIRPORT MOGADISHU-TOWN-HALL (RANGE UH-60A-1)`
VC3. `(> (SEA-TEMPORATURE MOGADISHU-PORT-ENTRANCE) 40)`
VC4. `(PLATOON-SIZED SECURITY-SQUAD-1)`

Expansion Template Structure

```
- HOSTAGE-RECOVERY-TO-A-POTENTIALLY-UNSTABLE-AREA
    - ADVANCED-RECON-OF-TARGET-AREA
        - RECON-SEAPORTS-IN-AREA
          - RECON-WITH-COVERT-GROUND-FORCE
              - SWIM-INSERTION-FROM-SUBMARINE
    - RESCUE-AND-RECOVER-HOSTAGES
        - HELICOPTER-INSERTION-ROPE
    - SITE-DEFENSE-LARGE-REACTION-FORCE
```

Figure 4. Violated Constraints and Plan Structure for the Selected Expansion

- drop the constraint *VC1*
- modify the *Helicopter* argument of the `DROP` task to be `UH-60L-1` rather than `UH-60A-1` to address the constraint *VC2* (i.e., the UH-60Ls have greater range than the UH-60As)
- drop the constraint *VC3*
- modify the *Force-Composition* argument of the `POSITION` task to be `SECURITY-PLATOON-1` (to address the violated constraint *VC4*)

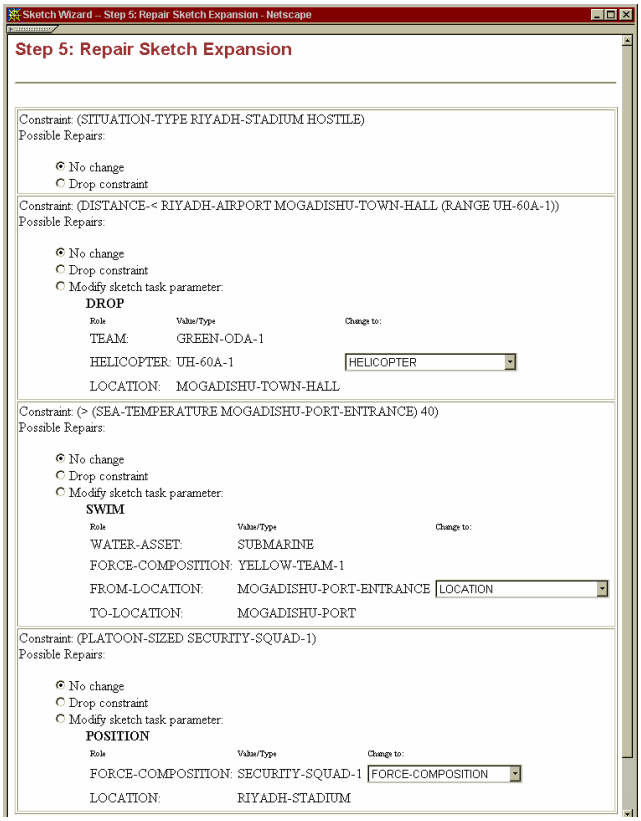

Figure 5. Sample Repair Options

Given a set of repairs, PASSAT attempts to validate the revised sketch. In this case, the repairs resolve the original problems but introduce a violation of the constraint `(COMBAT-EFFECTIVE SECURITY-PLATOON-1)`. This new problem can be repaired by changing the *Force-Composition* argument to be `SECURITY-PLATOON-2` (i.e., a platoon that has been certified ready for combat). Processing of this revised sketch yields a single expansion with no constraint violations. Figure 1 shows the insertion of that expansion for the original Hostage-Rescue task, with the sketch tasks highlighted in bold font. Constraints that the user chose to drop appear highlighted on the agenda, and are marked as completed but ignored. At this stage, the user could continue planning with the sketch result, using any of PASSAT's capabilities for interactive planning (e.g., applying templates, instantiating variables), or by providing a plan sketch for an unrefined objective.

Advice

In future work, we will extend PASSAT to enable a user to guide and control automated template expansion through the metaphor of *advice* [McCarthy, 1958]. Advice within PASSAT will express user recommendations for characteristics for the desired solution, thus limiting the set of allowed operations (human or automated) in constructing plans. Advice will be *heuristic*, capturing conditions that the user would like satisfied, but that can be relaxed if necessary.

PASSAT will monitor evolving plan content to identify violations of stated advice. Violations will lead to user notification, as well as the posting of appropriate planning task entries on the user's agenda. This work will build on our previous work on giving strategic advice to fully automated planners [Myers, 1996], with adaptations and extensions as required for use within a plan-authoring framework.

Usability Features

We have incorporated several features into PASSAT to facilitate its use within real applications.

Because the development of a plan may span several days or be interrupted by other duties, PASSAT offers the ability to save a plan and to restart it later. As PASSAT is further developed to support multiple planners working on a single plan, this facility will allow parallel efforts to be coordinated in a shared plan repository.

A planner may sometimes develop a part of the plan and realize that the initial idea will not work. The system currently allows the user to *undo* the steps in reverse order. In the future, the user will be able to back out of earlier steps without necessarily losing later, independent steps.

PASSAT is designed to reduce the chance of inadvertent errors. Strong typing for task, function, and predicate definitions enables the checking of inputs for consistency. If a processing error should occur in the system, the undo mechanism can provide recovery to a safe checkpoint.

Process Facilitation

PASSAT facilitates the user's plan-authoring process by helping the user track information that is important to the development of the plan. Process facilitation is supported primarily by two capabilities:

- A prioritized *agenda* of planning steps listing the decisions that the user must make to address problems or incompleteness in the current plan.
- A mechanism for identifying key *information requirements* implicit in the user's partial plan, and for directing the user's attention to relevant plan elements when new information arrives.

Agenda and Prioritization

PASSAT's agenda consists of the open planning steps facing the user given the current state of planning. By 'planning steps', we mean decisions and actions that the user makes in the process of developing the plan; these are distinguished from the activities that are part of the plan itself. PASSAT currently supports three types of planning step – *expand task*, *instantiate variable*, and *resolve constraint* – described earlier. The planning steps PASSAT displays in its agenda can be filtered by the user along several dimensions, including step type and completion status. The user can also sort the agenda along several dimensions, including step type, creation time, and alphabetical order. The filtering and sorting facilities can be especially useful for helping the user find a particular step on the agenda.

In real domains, the development of a plan can involve hundreds or even thousands of decisions. Correspondingly, PASSAT's agenda can grow quite long during the planning process. The system provides some basic mechanisms to control agenda growth – instantiating variables during template application, automatic calculation of constraints – and to control information overload in the agenda display – the aforementioned agenda filtering and sorting. However, even with these capabilities, the agenda can frequently reach a size that is overwhelming to the user. In the face of a large number of planning steps, we need a technique for keeping the human planner focused on the most important ones.

To deal with this problem, we have developed mechanisms for prioritizing the planning steps on the agenda, according to some notion of a step's importance to the planning process. Our approach has been to offer a suite of prioritization tools, from which the user may choose given the specific planning situation. Currently, PASSAT supports three prioritization approaches:

Predefined Each subtask, variable, and constraint in a template may be tagged with a qualitative priority (*high*, *medium*, or *low*), corresponding to the importance of making a decision about that entity (expanding the task, instantiating the variable, checking the constraint). Predefined priorities always take precedence over PASSAT's other prioritization methods in ordering the agenda display.

Commitment-based This approach prioritizes each planning step according to the degree that a decision will constrain the rest of the planning process, giving highest priority to the most constraining decisions. This criterion is especially useful in collaborative planning situations, where it is important to make decisions early when they will constrain the alternatives available to other planners. Our technique measures commitment as the expected number of future decisions eliminated by performing the step. We approximate this with a recursive formula that performs a lookahead search through the plan space. While we use some simple heuristics to reduce the size of the search, the current procedure is still reasonably expensive relative to PASSAT's other update calculations. As a result, the current implementation of commitment-based prioritization covers only tasks. In future work, we will investigate techniques for approximating the commitment level of a planning step more efficiently.

Experience-based In contrast to the commitment-based approach, which is an attempt to identify what the planner should do next based on some theoretical model of planning, the experience-based approach bases its prioritization on what real human planners have done first in the past. The experience-based prioritization technique stores preference histories of planning steps, and learns a preference function for them using the online learning algorithm of [Cohen et al., 1998]. Planning steps are indexed by the step type, the object name, and the 'call stack' of templates that created the object.

Other possible methods for deriving a step's priority include

- *Urgency-based:* prefer decisions that involve execution tasks that are scheduled to start soon.
- *Backtracking-based:* prefer decisions that are difficult to achieve. This is effectively the prioritization criterion of the Fewest Alternatives First strategy and related heuristics [Pollack et al., 1997] used in automated planning.
- *Depth-first:* prefer steps that derive from the steps most recently performed by the user. This approach assumes that the user wants to remain focused on one area of the plan before moving to another.
- *Breadth-first:* prefer steps that derive from the steps least recently performed by the user.

Information Requirements

In real-world planning, the human planner often makes decisions based on criteria that are too complex or vague to formalize in a predicate. These criteria are often based on external sources of information (e.g., reports, meetings). For example, a SOF planner may want to base his selection of a rendezvous point on an overall assessment of an intelligence report from the relevant region, though it may be virtually impossible to formalize the exact set of conditions the planner is looking for within that report. In a plan-authoring system, we want to be able to capture these criteria and information sources, and record the connection between them and the relevant elements of the plan. PASSAT accomplishes this through the use of *information requirements*.

In addition to specifying the method for expanding a task, a template may also include one or more information requirements. An information requirement specifies a monitoring condition on an information source that may be useful for determining the applicability of the template, for selecting variable instantiations, or for resolving the template's constraints.

Currently, information requirements are used in PASSAT to make explicit to the user the connection between plan elements (e.g., *variables*, *constraints*) and information sources. When a planner activates an information requirement in a template, the system creates a link between the information described in the information requirement and an element or elements in the plan. When the information arrives, PASSAT calls the planner's attention to the relevant plan element by creating a high-priority item on the agenda to revisit that element. PASSAT's current method of detecting when information has arrived is to be told explicitly by a user, but one could imagine more sophisticated automated *sentinels* that would, for example, monitor data sources (e.g., Web pages, databases) for specific updates.

For example, a user planning a SOF mission may make a tentative assignment to a variable `?RENDEZ-POINT` based on the sketchy information available to him. At the same time, he may activate an information requirement representing an intelligence report on the region in question and attach it to the variable `?RENDEZ-POINT`. When the intelligence report comes in, PASSAT will notify the planner by putting the Instantiate Variable step for `?RENDEZ-POINT` back on the active agenda, giving it a high priority, and highlighting the element on the planner's agenda display.

Related Work

In its effort to increase relevance to real problems, the field of AI planning has recently produced a number of more human-centric technologies that incorporate both interactive and automated planning capabilities. Work in this area has progressed on two fronts: (a) the incorporation of more sophisticated reasoning into simple plan specification tools, and (b) the addition of interactive and mixed-initiative capabilities into existing automated planning systems. The first category includes systems such as the SOFTools Temporal Plan Editor, APGEN, and INSPECT. Examples in the second category include O-Plan, Heracles, and TRIPS.

The SOFTools Temporal Plan Editor [GTE, 2000] supports the graphical specification of a collection of activities on a series of timelines; its automated capabilities are limited to simple syntactic checking (e.g., action start times precede end times). The APGEN system [Maldague et al., 1997] provides a timeline-oriented interface for creating mission sequences as well as automated validation of predefined flight constraints. There is currently an effort under way to link APGEN to the RAX-PS planner [Jonsson et al., 2000] to enable the automated synthesis of plans. The resulting system will facilitate user-driven exploration of options, as automation enables candidate plans to be generated rapidly. INSPECT [Valente et al., 1999] provides an interactive planning environment in which users can create plans by drawing on predefined knowledge bases of planning operators. A knowledge-based critic looks for problems in user-formulated plans, both syntactic and (in limited cases) semantic.

Within Heracles [Knoblock et al., 2001], a user can construct plans by interactive selection and instantiation of predefined HTN-style templates. Heracles provides constraint reasoning that facilitates the planning process by focusing users on choices that are guaranteed compatible with earlier decisions. Plans must instantiate the predefined templates, thus preventing users from exploring 'out of the box' solutions. The TRIPS system [Ferguson and Allen, 1998] provides a dialog-based interface to a temporal planner that enables users to interactively guide the construction and execution of a plan through a cooperative, mixed-initiative effort.

O-Plan was developed initially as a fully automated HTN planning system but has been modified to incorporate interactive capabilities such as user support for operator selection and variable instantiation [Drabble and Tate, 1995], and human-driven exploration of multiple courses of action [Tate et al., 1998]. PASSAT lacks somem of the automated planning capabilities within O-Plan (i.e., there is no infrastructure to support automated search with

backtracking), being focused instead on more human-centric planning methods (interactive planning, sketching, advisability). O-Plan contains a task agenda similar to that in PASSAT, but no prioritization methods. Furthermore, it does not include information requirements, or capabilities related to sketching or advice.

Conclusions

Our long-term objective for PASSAT is to provide a planning environment that covers the range from purely interactive through mixed-initiative to user-controllable automated planning capabilities. At all times, automation would be readily controllable and understandable by a human planner, enabling humans to determine when automation is used, to control how automation applies, and to validate or override any automated decisions.

PASSAT currently provides a strong base of interactive, template-based plan authoring and robust sketch-based planning. Our main next steps on PASSAT are (a) to increase the flexibility of the interactive planning, and (b) to implement the advisability module for imposing high-level constraints on a plan that can be validated automatically.

Acknowledgments. This work was supported by DARPA under Air Force Research Laboratory Contract F30602-00-C-0058.

References

Allen, J. F. (1984). Towards a General Theory of Action and Time. *Artificial Intelligence* 23.

Cohen, W. W., Schapire, R. E., and Singer, Y. (1998). Learning to Order Things. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds). *Advances in Neural Information Processing Systems*, The MIT Press.

Currie, K., and Tate, A. (1991). O-Plan: The open planning architecture. *Artificial Intelligence*, 32(1).

Drabble, B., and Tate, A. (1995). O-Plan Mixed Initiative Planning Capabilities and Protocols, Technical Report, University of Edinburgh.

Erol, K., Hendler, J., and Nau, D. (1994). Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Ferguson, G., and Allen, J. (1998). TRIPS: Towards a Mixed-Initiative Planning Assistant. In *Proceedings of the AIPS Workshop on Interactive and Collaborative Planning*.

GTE. (2000). *SOFTools User Manual*.

Jonsson, A. K., Morris, P. H., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth International Conference on AI Planning Systems*.

Karp, P. D., Myers, K. L., and Gruber, T. (1995). The Generic Frame Protocol. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*.

Knoblock, C. A., Minton, S., Ambite, J. L., Muslea, M., Oh, J., and Frank, M. (2001). Mixed-initiative, Multi-source Information Assistants. In *Proceedings of the International World Wide Web Conference*.

Maldague, P., Ko, A. Y., Page, D. N., and Starbird, T. W. (1997). APGEN: A Multi-Mission Semi-Automated Planning Tool. In *Proceedings of the 1st NASA Planning and Scheduling Workshop*.

McCarthy, J. (1958). Programs with Common Sense. *Symposium on the Mechanization of Thought Processes*.

Myers, K. L. (1996). Strategic Advice for Hierarchical Planners, pp. 112-123. In L. C., and S. C Aiello, J. Doyle. Shapiro (Eds): *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, Morgan Kaufmann Publishers.

Myers, K. L. (1997). Abductive Completion of Plan Sketches. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, AAAI Press.

Pollack, M. E., Joslin, D., and Paolucci, M. (1997). Flaw Selection Strategies for Partial-Order Planning. *Journal of Artificial Intelligence Research* 6, 223-262.

Tate, A. (1977) Generating Project Networks, in Proceedings of the Fifth International Joint Conference on Artificial Intelligence.

Tate, A., Dalton, J., and Levine, J. (1998). Generation of Multiple Qualitatively Different Plans. In *Proceedings of the Fourth International Conference on AI Planning Systems*, Pittsburgh, PA.

Valente, A., Blythe, J., Gil, Y., and Swartout, W. (1999). On the Role of Humans in Enterprise Control Systems: The Experience of INSPECT. In *Proceedings of the DARPA-JFACC Symposium on Advances in Enterprise Control*.

Wilkins, D. E. (1993). Using the SIPE-2 Planning System: A Manual for Version 4.3, Artificial Intelligence Center, SRI International, Menlo Park, CA.

Toward a Theory of Qualitative Reasoning about Plans

Karen L. Myers
AI Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com

Abstract

Automated planning algorithms embody a theory of causality grounded in linking enabling effects of actions or initial world conditions to preconditions of subsequent actions. This model has several drawbacks when applied in the context of mixed-initiative planning. First, it requires comprehensive causal models that describe for every action its full set of preconditions and postconditions; in many application domains, such models will not be available. Second, the causal link approach does not include several forms of intraplan relations that a human may wish to document and reason with.

We present a *qualitative* approach to reasoning about plan structure that is designed for mixed-initiative plan development. We define a set of plan relations that characterize key interactions among plan components, and an accompanying calculus for reasoning qualitatively about the effects of changes on a plan. We argue that such an approach is better suited to mixed-initiative planning than is the standard causal link method.

1 Introduction

As the AI planning community turns increasingly to realistic problem domains to motivate its work, there has been a growing recognition of the need for mixed-initiative planning techniques. This need is driven by two concerns. First, full automation is inappropriate for many application domains. In particular, most users want to be involved in the planning process, both to influence the types of solution that are produced and to develop an understanding of the process by which a solution was formulated. Second, the cost of formulating the correct and comprehensive background theories required by automated planners is prohibitive. These theories consist of causal models that describe for every action its *preconditions* (i.e., the conditions under which it could be applied) and its *postconditions* (i.e., the conditions that result from execution of the action). Most planning knowledge bases to date provide only poor approximations to realistic models of activity. Furthermore, formulation of even these simplified models remains an art that requires highly trained specialists in AI planning and knowledge representation. Mixed-initiative planning systems can reduce the knowledge requirements by relying on the user to provide information when necessary.

Most work to date on mixed-initiative planning has assumed a comprehensive set of background knowledge that can be used by the system to validate interactions with the user (e.g., (Allen and Ferguson, 2002; Kim and Blythe, 2003)). We are interested in a model of mixed-initiative planning that does not require complete reliance on predefined knowledge bases. In particular, we view plan development as a cooperative problem-solving process in which both the system and the user contribute knowledge during a planning session.

One consequence of this collaborative model is that the system can no longer guarantee generation of a standard causal link structure for the plan (Weld, 1999), as not all required knowledge would be resident within the system's knowledge base. Causal structures enable proof of the 'correctness' of a plan, meaning that simulated execution of the actions in the plan from the initial state yields a state that satisfies stated objectives. Additionally, they provide the means to answer the following questions (Kambhampati and Hendler, 1992), which are important for both automated and collaborative planning:

A. What role does a given action, constraint, or assumption play in a plan?
B. What impact would a given change have on a plan?

For these reasons, the planning community has been reluctant to move toward models where causal structures are not a by-product of planning.

To obtain a full causal structure for a plan when background models are incomplete or unavailable, a human planner would have to annotate plans. There are two problems with such an approach. First, supplying a complete set of causal annotations would be a time-consuming and laborious task. Second, the formulation of causal link justifications for activities is a highly technical skill that is beyond most users.

We believe that much of the value of these complex causal models can be attained through simpler, *qualitative* models that capture commonsense notions of intraplan relationships. In this paper, we propose a user-centric model for causal structure that is

grounded in *qualitative* relationships among plan objects. The basic idea is to trade some of the detail and precision of the formal causal models for a simpler approach that is easier both to formulate and apply. This simpler approach would still enable answers to questions (A) and (B) above, although in qualitative rather than quantitative terms. Furthermore, as we argue below, the models of causality embraced by the AI planning community are unnecessarily narrow because of their evolution from the structures required for automated planning. In particular, they do not capture certain notions of plan dependency that are important for mixed-initiative planning systems.

An interesting parallel can be drawn between plan development and other synthesis tasks such as mechanical design. Commercial design systems support human design activities by providing automation for low-level, tedious tasks. A designer will generally not document every step/component within a design. Indeed, studies have shown that humans are resistant to providing full rationale information as part of the design process, due both to the substantial time commitment required and the changes in work style that result (Carroll and Moran, 1991; Conklin and Yakemovic, 1991). However, designers track key dependencies and requirements that they believe will be important in understanding and maintaining a design downstream. We believe that this user-driven approach should also be the objective for documenting causal structure within mixed-initiative planning technology.

The remainder of the paper is organized as follows. Section 2 presents our qualitative model of plan relations. Section 3 provides an example that contrasts the use of standard causal link structures with our qualitative approach. Section 4 defines a calculus for reasoning qualitatively about plan changes, while Section 5 introduces conditions of coherence for a set of qualitative causal relations. Throughout, we draw on examples from a simplified version of a noncombatant evacuation operation (NEO) domain that was developed for use within the PASSAT mixed-initiative planning system (Myers et al., 2002).

2 Qualitative Plan Model

Ideally, a system that reasons about plan structure should combine both causal links and qualitative information about plan relationships. We define a candidate set of relations to support qualitative reasoning about the effects of plan changes that contains relations of both types.

2.1 Plan Elements

Our model of a plan contains three types of element:

Action: an activity that can be undertaken
Effect: a condition (either to be achieved, the expected result of executing an action, or a property of the initial world state)
Parameter: an argument to an action or condition

We use the symbol *Obj* (i.e., plan object) to denote an arbitrary plan element from any of the above types.

A plan relation is defined between a *source object* and a *target object* and is represented using the syntax:

Reln: Source-Obj → Target-Obj

2.2 Causal Link Relation

The *causal-link* relation, which is the standard relation within most automated planning systems, indicates that the source effect is a necessary condition for the target effect.

Causal-link: {Effect} → { Effect }

Figure 1. Causal Link Relation

2.3 Qualitative Relations

Figure 2 summarizes our qualitative plan relations. Broadly speaking, the qualitative relations can be separated into two categories: *temporal* (QR1 – QR3) and *logical* (QR4 – QR5).

QR1	*Precedes: {Action | Effect} → {Action | Effect}*
QR2	*Necessary-for: {Action | Effect} → {Action | Effect}*
QR3	*Supports: {Action | Effect} → {Action | Effect}*
QR4	*Parameter-dependence: {Parameter} → {Parameter}*
QR5	*Condition-dependence: {Effect} → {Action | Effect | Parameter}*

Figure 2. Qualitative Plan Relations

2.3.1 Qualitative Temporal Relations

The qualitative temporal relations capture the notion that a given action or effect in a plan must precede some other action or effect. We consider three types: *precedes*, *necessary-for*, and *supports*.

The *precedes* relation captures the notion that the specified source action or effect should occur before the specified target action or effect, without providing any indication of why. This type of relation can be used to capture a preference for performing activities in some designated order when there is no necessary reason for that order. For example, consider the actions of preparing an evacuation site and flying evacuees to the evacuation site. Although it would be possible to perform those actions in parallel, a given planner may have a preference for completing the preparation prior to the start of the airlift of the evacuees, possibly to enable a delay of the airlift in the event of problems with the preparation.

The *necessary-for* and *supports* relations specialize the *precedes* relation to capture semantic motivations for the ordering relation. *Necessary-for* captures the notion that a given action or effect must occur before a designated action or effect in order to enable the target plan element. For example, it would be *necessary-for* evacuees to be marshaled to an assembly point before they could be loaded onto an evacuation aircraft. In essence, the *necessary-for* relation constitutes a qualitative abstraction of the *causal-link* relation. Changes could impact plan objects linked by a *necessary-for* relation in two ways. First,

delays to necessary activities will propagate. Second, failure of a task that is *necessary-for* another task would likely jeopardize the latter.

The *supports* relation indicates that the source action or effect contributes to the target action or effect in some noncritical way,. For example, a patrol mission may provide additional support to a given evacuation activity, without being essential to its undertaking. Hence, if the aircraft performing the patrol were redirected to support a different action, the evacuation process should not be jeopardized. Source objects for *supports* relations correspond to 'redundant' actions or effects that, while unnecessary, lead to improved plan robustness or quality. While such redundancy is considered good practice in human-authored plans, causal link planners explicitly prohibit redundancy by imposing conditions of minimality on a plan's causal structure.

2.3.2 Qualitative Logical Relations

The qualitative logical relations QR4 and QR5 capture the idea that there is some form of dependency between the source and target elements such that a change to the source could impact the target. However, the nature of that relationship is not captured precisely in terms of a deductive specification or mathematical formula. Such situations arise frequently in planning situations, where many factors that influence decision-making are problematic to formalize. These factors could include conditions that are too complex to codify (i.e., a form of the *qualification problem* (McCarthy 1977)) or subjective preferences that vary among human planners.

For example, the choice of assembly point in an evacuation plan will necessarily impact the type of aircraft that can be used for transporting evacuees (e.g., a small helicopter may be necessary for evacuation from an embassy, while a larger aircraft could be used at a football stadium). However, there is no hard-and-fast rule for determining what type of aircraft should be used for a particular location.

Qualitative logical relations can be designated between plan parameters (QR4), or between plan effects and any type of plan component (QR5). The relationship between the choice of assembly location and transport aircraft in the example above corresponds to a *parameter-dependence* relation, while the relationship between the security level and choice of assembly point corresponds to a *condition-dependence* relation.

We note that the qualitative logical relations could be made 'quantitative' by associating definite constraints with them. For the *parameter-dependence* relation, these constraints would be in the form of a set of equations linking the two parameters. We introduce the term *parameter-constraint* relation to refer to this specialization of the *parameter-dependence* relation. One of the most valuable forms of *parameter-constraint* relation would be an equality constraint indicating that two planning variables must necessarily be instantiated to the same value (such constraints are sometimes referred to as *codesignation* constraints). A comparable *condition-constraint* relation could similarly be defined.

2.4 Properties of the Model

The qualitative model trades the precision of exhaustive causal links for simplicity and ease of specification. Indeed, there is a natural abstraction from an exhaustive causal link structure to 'corresponding' qualitative models that involves replacing every *causal-link*

relation with a *precedes* or *necessary-for* relation. We refer to a plan transformed in this manner as a *qualitative abstraction* of the original.

While the qualitative relations provide less precision than a full set of causal links, they offer two key advantages. First, they are simpler and more intuitive to specify, thus making them better suited for use in a mixed-initiative planning environment. Second, there are relationships that can be modeled in the qualitative framework that are not supported by causal links or logical constraints associated with operator preconditions. In particular, the supports relation enables the description of a connection between plan objects that is not essential for plan correctness (as described above). As well, the precedence relation allows the expression of ordering information independent of causality. Furthermore, the condition-dependence and parameter-dependence relations enable ill-defined connections between plan objects to be expressed; this ability is useful when the precise logical or mathematical relationship is either not known or not easily formalizable, yet there is still a desire to document some relationship between them.

2.5 Sources

Our motivation for defining qualitative relations is to enable reasoning about plan changes within a mixed-initiative planning environment that combines human and automated planning skills. Within this context, several sources would contribute to the set of causal relations for a given plan. First, background knowledge could capture both causal link and qualitative causal relations for individual actions or within 'standard operating procedures' that arise frequently in practice (i.e., in the style of hierarchical task networks (Erol et al. 1994). Second, the human planner could contribute additional relations. In the future, some sort of learning mechanism could be applied to hypothesize qualitative relations from a user-authored plan, yielding a baseline that a user could then modify (e.g., along the lines proposed by (El Fattah 2003)).

3 Example: Using the Qualitative Relations

To illustrate the use of the qualitative relations for planning, consider the plans in Figure 3 and Figure 4 for a simple evacuation operation. These plans are designed to achieve the goal conditions *Prepared(Camp1)* and *At(Evacuees Camp1)* based on the operators defined in Figure 5. Each of the plans includes conditions from the initial state upon which actions and effects in the plan depend, as well as the effects that constitute the desired goal state. To simplify reference, each action in the plan is labeled with a unique identifier (e.g., *N1*).

The plan in Figure 3 corresponds to a solution that an automated causal link planner might produce for this problem. It includes a full causal link annotation that matches each action precondition to an earlier effect in the plan.

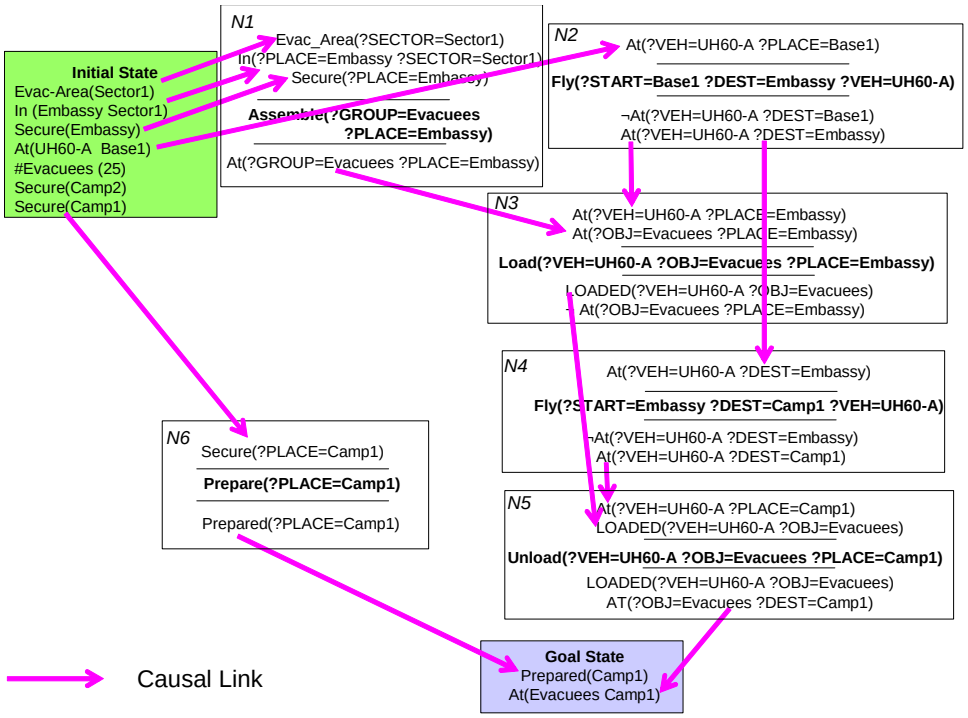

Figure 3. Evacuation Plan with a Complete Set of Causal Link Relations

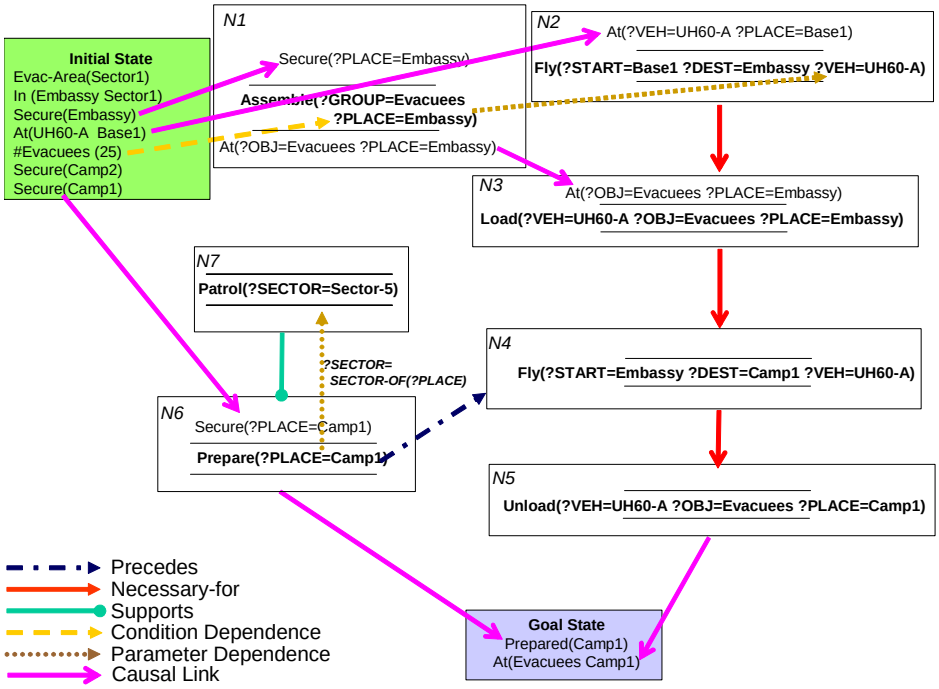

Figure 4. Evacuation Plan with Qualitative and Causal Link Relations

The plan in Figure 4 represents a solution that a human planner might construct using some kind of plan authoring tool. It contains all of the actions in Figure 3 plus a *Patrol* action (*N7*) that provides additional security for the sector to which the evacuees will be moved. According to the logic of the domain operators, this action is redundant because it does not establish any effects that are required within the plan. However, it is typical for human planners to build such redundancy into plans to provide additional safeguards in the face of unexpected events.

The plan in Figure 4 also contains a candidate set of both casual link and qualitative relations that document what the user might view as the key dependencies within the plan. The main differences between this hybrid set of plan relations and the causal link relations in Figure 3 are as follows:

- Causal-link relations in Figure 4 are limited to dependencies on initial state conditions that might be expected to change and hence may require modifications to the plan (e.g., the security of key locations and the position of the vehicle to be used for transporting the evacuees) and important intermediate effects of action (e.g., the evacuees remain at the embassy until they are loaded onto the transport vehicle). Static initial conditions and unimportant intermediate effects of actions have been omitted.
- The hybrid annotation replaces certain of the *causal-link* relations with qualitative *necessary-for* relations, indicating that it is essential for the source activity to precede the target activity in the plan but without documenting the effects that link the actions. From the user's perspective, these effects are obvious (e.g., the aircraft has to be loaded before it can be unloaded) and so documenting them explicitly is of little value.
- The qualitative annotations include a *precedes* relation from node *N6* to node *N4*, indicating a (noncausal) preference for ordering those two actions. This ordering is not necessary for the plan to succeed.
- A *condition-dependence* relation has been added from the predicate *#Evacuees(25)* in the initial world state to the parameter *?PLACE* in *N1* where the evacuees are to be assembled. This relation reflects the fact that the choice of assembly location is dependent on the number of evacuees; should the number change, the choice may need to be revisited.
- A *parameter-dependence* relation has been added from *?PLACE* in *N1* to *?Veh* in *N2* and a *condition-dependence* relation added from *#Evacuees(25)* in the initial world state to *?Veh* in *N2*. These relations show that the choice of vehicle depends on both the number of evacuees and their assembly location, although the precise nature of the dependency is unknown.
- Figure 4 contains a *supports* relation from *N7* to *N6*, documenting that the *Patrol* action is being performed in service of the *Prepare* action. No comparable link between these nodes is possible in the causal link view because there is no enabling relationship between effects produced by *N7* and required by *N6.* A *parameter-dependence* relation has also been added from the *?PLACE* parameter in *N6* to the *?SECTOR* variable in *N7* indicating that the choice of patrol area depends on the evacuation site. In this case, the relationship could be expressed

algebraically by adding the constraint *?SECTOR=SECTOR-OF(?PLACE)* to the *parameter-dependence* relation, thus yielding a *parameter-constraint* relation.

Action: *Assemble(?GROUP ?PLACE)*
Preconditions: *Secure(?PLACE), Evac-Area(?SECTOR), In(?PLACE ?SECTOR)*
Effects: *At(?GROUP ?PLACE)*

Action: *Fly (?START ?DEST ?VEH)*
Preconditions: *At(?VEH ?START)*
Effects: *At(?VEH ?DEST), ¬At(?VEH ?START)*

Action: *Load(?VEH ?OBJ ?PLACE)*
Preconditions: *At(?VEH ?PLACE), At(?OBJ ?PLACE)*
Effects: *Loaded(?VEH ?OBJ), ¬At(?OBJ ?PLACE)*

Action: *Unload(?VEH ?OBJ ?PLACE)*
Preconditions: *At(?VEH ?PLACE), Loaded(?VEH ?OBJ)*
Effects: *At(?OBJ ?PLACE), ¬Loaded(?VEH ?OBJ)*

Action: *Prepare(?PLACE)*
Preconditions: *Secure(?PLACE)*
Effects: *Prepared(?PLACE)*

Action: *Patrol(?PLACE)*
Effects: *Prepared (?PLACE)*

Figure 5. Evacuation Planning Operators

4 Qualitative Calculus

The qualitative calculus determines the impact that different types of change can have on a given plan. Here, we consider only changes to plan objects (parameters, actions, effects). More generally, changes to qualitative and casual link relations should also be considered.

4.1 Impact of Core Plan Changes

We consider the space of changes to plan elements listed in Figure 6. Each entry in the figure lists a type of plan change along with the class of plan elements to which the change applies.[1]

[1] Within our framework, the addition of a plan object on its own does not impact qualitative effects, as the new object will not (yet) be linked to any other plan object. Since this version of our framework does not consider changes to plan relations, we ignore plan object addition.

Change-Parameter: {Parameter}
Cancel: {Action | Effect}
Delay: {Action | Effect}
Move-forward: {Action | Effect}
Change-Truth-Value: {Effect}

Figure 6. Plan Changes

Plan changes can have a range of effects on plan elements linked by qualitative and casual-link relations. Our qualitative model considers a set that includes temporal, correctness, and minimality effects. In terms of temporal effects, *delay* indicates that a plan element may be moved further back in time, while *move-up* indicates that a plan element may be moved forward in time. In terms of correctness, *disable* indicates that the plan element is no longer 'supported' within the plan. In terms of minimality, *obviate* indicates that the plan element may no longer be needed in the plan (e.g., it no longer plays a role within the plan). The effect *affect* indicates that there may be some impact but that no additional conclusions about the nature of the impact can be determined.

In general, qualitative reasoning will enable inference of potential rather than definite impact on a plan. For this reason, we can categorize qualitative effects according to the two modalities of *necessity* and *possibility*, indicating whether the effect necessarily occurs or only may occur. Because of the loss of precision inherent to the qualitative models, it is generally difficult to establish necessary effects. However, consideration of explicit quantitative information (see Section 4.2) does allow inference of necessary effects in some cases.

Plan changes can have impacts that flow along both 'directions' of a plan relation. In particular, changing the source element in a plan relation could impact the target element (i.e., the *forward* direction) while changing the target element could impact the source element (i.e., the *backward* direction).

Below, we consider the direct impact of each type of change on objects related by the qualitative relations QR1 – QR5, as well as the *causal-link* and *parameter-constraint* relations. Figure 7 summarizes the results.

4.1.1 Cancel

The *cancel* operation deletes an action or effect from a plan. Equivalently, it can be interpreted as having an action that fails at execution time or an effect that is violated. A *cancel* operation can impact only *necessary-for* and *causal-link* relations in the forward direction. In the event that the *Source-Obj* of a *necessary-for* or *causal-link* relation is canceled, the *Target-Obj* would have a necessary enabling effect violated and hence would be disabled. Cancellation of the *Target-Obj* could potentially obviate the need for the *Source-Obj* for *necessary-for*, *supports*, and *causal-link* relations. However, that would be the case only if the *Source-Obj* was not also the source for a *necessary-for*, *supports*, *condition-dependence*, or *causal-link* relation for a different plan element.

4.1.2 Change-Parameter

The *change-parameter* operation modifies an argument to an action or effect in a plan. Such a change can impact elements related by *parameter-dependence* or *parameter-constraint* relations. In the case where the source is changed and the target is an

unbound parameter (similarly, the target is changed and the source is an unbound parameter), there will be no qualitative impact. Hence, we consider only instantiated parameters here and in Figure 7.

For the *parameter-dependence* relation, a change to *Source-Obj* could necessitate a change to *Target-Obj*, although the precise nature of the change would not be determinable without additional information. Thus, the impact of a change in the forward direction is listed as *affect* in Figure 6. In contrast, the *Source-Obj* is independent of the value of *Target-Obj*, so changes to the latter will not affect the former. To illustrate, consider a *parameter-dependence* relation from the number of evacuees in a NEO operation (*Source-Obj*) to the choice of assembly site (*Target-Obj*). In the event that there is an increase in the number of evacuees, a planner may need to reconsider the chosen assembly site (since it may not be large enough to accommodate the enlarged set of people). However, changing the assembly site would not involve any reconsideration of the number of evacuees involved (since that number is beyond the control of the planner).

The impact on a *parameter-constraint* relation is similar in the forward direction but differs somewhat in the backward direction. In particular, the *parameter-constraint* relation is necessarily 'bidirectional', being grounded in an actual algebraic constraint. For this reason, changes to the *Target-Obj* of such a relation would yield a possible change in the *Source-Obj*; thus, the qualitative impact of such a change would be *affect.* To illustrate, consider a *parameter-constraint* relation between the parameters *?X* and *?R* defined by the equation $?X = \pi * ?R^2$. In this case, changing *?X* or *?R* will clearly impact the value of the other.

FORWARD IMPACT	Delay	Move-up	Cancel	Change Param	Change TV
Precedes	*delay*	*	*	*	*
Necessary-for	*delay*	*	*disable*	*	*
Supports	*delay*	*	*	*	*
Parameter-dependence	*	*	*	*affect*	*
Condition-dependence	*	*	*	*	*disable*
Causal-link	*delay*	*	*disable*	*	*disable*
Parameter-constraint	*	*	*	*affect*	*

BACKWARD IMPACT	Delay	Move-up	Cancel	Change Param	Change TV
Precedes	*	*move-up*	*	*	*
Necessary-for	*	*move-up*	*obviate*	*	*
Supports	*	*move-up*	*obviate*	*	*
Parameter-dependence	*	*	*	*	*
Condition-dependence	*	*	*	*	*obviate*
Causal-link	*	*move-up*	*obviate*	*	*obviate*
Parameter-constraint	*	*	*	*affect*	*

Figure 7. Summary of Direct Qualitative Effects for Forward and Backward Impact

4.1.3 Change-Truth-Value

The *change-truth-value* operation involves modifying the truth-value of a designated effect, either from *true* to *false* or *false* to *true.* Such changes could potentially impact plan elements linked by a *condition-dependence* relation. In the forward direction, change to the source effect of such a relation threatens the viability of the target object, hence the qualitative impact is *disable*. Change to the target element could eliminate the need for the source effect in the event, specifically when the target element is an effect and the change establishes the truth of the effect. In this case, the source object may no longer be required in the plan, provided that no other plan element depends on it (i.e., the effect is the source object for some other qualitative relation). For this reason, the qualitative effect is listed as *obviate*.

The same qualitative effects apply for the *causal-link* relation, as causal links are a specialization of the *condition-dependence* relation.

4.1.4 Delay

A delay operation impacts the three temporal qualitative relations *precedes, necessary-for,* and *supports*, while exhibiting similar behavior for the qualitative relation *causal-link*. We consider the impact of a delay operation with the assumption that each plan element has an explicit representation of timing information for the occurrence of that plan element. Such information could be in the form of a timepoint for instantaneous actions, or a time window for actions with duration.

Simple qualitative effects can be established for delay operations without considering anything beyond the individual qualitative relation. *Target-Obj* will possibly be delayed when *Source-Obj* is delayed (and necessarily when the time for Target-Source is greater or equal to the time assigned to *Target-Obj*). A delay to *Target-Obj* will have no impact on *Source-Obj*.

4.1.5 Move-Up

The effects of the move-up operation are the inverses of those of the Delay operation. Specifically, a move-up operation will impact only the temporal qualitative relations (*precedes, necessary-for, supports*) and the *causal-link* relation, and only in the backward direction. The qualitative impact will be a possible *move-up* of the source-object in such relations.

4.2 Calculus Specializations

By specializing information about qualitative relations, additional inferences can be drawn regarding the impact of plan changes. In particular, the availability of quantitative information regarding the constraints linking parameters can lead to more detailed modeling of effects.

To illustrate, consider the case where there is information available about temporal slack within a plan. Consider a qualitative temporal relation *Source-Obj* → *Target-Obj* such that the constraint *TIME(Target-Obj) - TIME(Source-Obj)* < *D* defines a temporal buffer between the times for the two plan elements. If the TIME assignments for both plan elements are defined and a Delay operation is applied to *Source-Obj* such that the constraint is violated, then necessarily the *Target-Obj* must be delayed. Similarly, if a

Move-Up operation is applied to *Target-Obj* such that the constraint is violated, then the time of *Source-Obj* must also be moved forward in time.

4.3 Change Propagation

Figure 7 summarizes the direct qualitative effects that result from various types of plan change. Such direct effects can be propagated across qualitative rules to establish the full qualitative impact of a change on a given plan. Propagation involves repeatedly treating each new effect as a 'change' and then determining what new direct effects are triggered by those changes.

One general concern with such propagation methods is that they can lead to infinite reasoning cycles. That is not the case within our calculus. Firstly, the impact of changes for the Delay, Move-up, Cancel, Change-TV operations are all *directionally focused*, meaning that while they can trigger chains of effects, those chains will all be either in the forward or the backward direction. If we limit ourselves to cases where relations are acyclic (see the discussion on *coherence* for qualitative relations in Section 5 below), then the length of these chains is bounded by the length from the modified plan object to the end or start (depending on direction) of the plan. While there may be multiple chains for each change, the total number is bounded by the number of plan nodes.

The Change-Param operation is similarly directionally focused provided that there are no *parameter-constraint* relations involved. The *parameter-constraint* relation, however, can lead to effects that trigger reasoning that moves both forward and backward. Because the impact of the changes is limited to a single effect – namely, *affect* – it is clear that the propagation can be halted on any node for which that effect has already been generated in response to a given plan change. Hence, the reasoning is necessarily finite.

4.4 Example: Reasoning with Qualitative Effects

For the plan in Figure 4 with qualitative and causal link relations, we consider the qualitative impact of three changes.

- Move up by one hour the start of the FLY action in *N4*.

  There is a path of qualitative temporal relations from every action node in the plan except *N5* to *N4*. Thus, the propagation of direct qualitative effects of this change yields a *move-up* effect for each node except *N5*.

- Revise the initial state predicate *#Evacuees(25)* to *#Evacuees(50)*.

  The direct qualitative effect of that change would be an *affect* inference for the assembly site (i.e., *?PLACE* in *N1*); as a result, the value for this parameter may need to change. Propagation of this direct effect would further lead to an *affect* inference for the choice of vehicle used to transport the evacuees (i.e., *?VEH* in node *N2*) by virtue of the *parameter-dependence* link from *?PLACE* in *N1*to *?VEH* in node *N2*.

- Eliminate *N7* from the plan.

  No additional effects result, as no other part of the plan critically depends on this node.

Another interesting case to consider is changing the goals of the plan to be *Prepared(Camp2)* and *At(Evacuees Camp2)*. With the relations as stated in Figure 4, no qualitative effects would be deduced. Clearly, however, the ideal would be to determine that the *?PLACE* locations in nodes *N5* and *N6* should both change to *Camp2*, and further impacts then assessed.

Standard causal links support this type of reasoning, as they directly connect the changed parameters to the impacted variables. This behavior could be attained within the qualitative framework provided that the user added *parameter-dependence* relations, or *parameter-constraint* relations with explicit equality constraints among such codesignating parameters. Realistically, however, it is unlikely that users will want to document their plans at that level of detail.

One simple solution to this problem is to augment the qualitative reasoning calculus with a mechanism that responds to the changing of a parameter value from *v1* to *v2* by marking all other occurrences of *v1* in the plan (as well as in the goal or initiial conditions linked to any portion of the plan) as having a possible effect *affect* as a result of the change. In this example, the mechanism would then identify the *?PLACE* parameters in nodes *N5* and *N6* as having the effect *affect,* as well as the initial state condition *Secure(Camp1).*

By modifying the *?PLACE* parameters to be bound to *Camp2*, the user would in turn trigger recalculation of the parameter *?SECTOR* in node *N7* via the *parameter-dependence* relation with constraint *?SECTOR=SECTOR-OF(Camp2).* The marking of the initial state condition *Secure(Camp1)* as having the effect *affect* would notify the user of the need to ensure the revised condition *Secure(Camp2).* Because that condition is already true in the initial world state, no further actions are needed to address the goal changes.

5 Coherence of Qualitative Plans

Causal link structures provide a formal basis for determining the validity of a plan, by showing that the actions in the plan lead to a state where the goal conditions are satisfied, and that the preconditions of each action are enabled by effects of some earlier action or the initial world state. Minimality conditions can also be established by showing that it is not possible to remove any set of actions from the plan while continuing to satisfy the correctness criteria. The causal links within a correct and mimimal plan can thus be viewed as necessary and sufficient for the plan's validity.

This model of validity does not make sense for plans whose causal structure is documented in terms of a partial set of causal link and qualitative relations. First of all, the user would supply only those relations that he believes are important, with others remaining implicit in his internal model of the plan. Second, the plan cannot be proven 'correct' as there is no background model to underpin the validation. Third, the user may have supplied additional relations that transcend the enablement conditions inherent to causal link plan validation.

Instead, focus must shift to a weaker notion focused on ensuring that the relations are consistent. To this end, we introduce the notion of *coherence* for a plan with qualitative causal annotations.

Definition [Coherence Conditions]:
A set of actions A linked by qualitative relations Q is said to be *coherent* iff the following two conditions hold.

- **Temporal Acylicity:** The subgraph formed by the *Precedes*, *Supports*, and *Necessary-for* relations is acyclic.
- **Modality Exclusivity:** If there is a path of *Supports* relations linking plan elements P1 and P2, then there is no path of *Necessary-for* relations linking P1 and P2.

The temporal acylicity condition ensures that the plan does not have temporal loops. The modality consistency condition ensures that a relationship between plan elements is not declared as being both necessary and nonnecessary.

6 Summary

Qualitative reasoning about the effects of changes on plans has several virtues over standard logical/deductive approaches. First, qualitative reasoning does not require comprehensive and correct causal theories. While qualitative inferences can be drawn from incomplete models, however, more complete models will yield more informative results. Second, qualitative relations are simpler and more intuitive to define, making it possible for users to annotate plans with qualitative relations that reflect their specific needs and interests. In contrast, traditional deductive approaches require sophisticated models that have proven to be difficult for users to formulate. Third, qualitative models include relationships that do not require complete formalization of concepts, making them usable in situations where precise dependencies among plan elements cannot be articulated.

One of the problems with the qualitative framework defined here is that it is derived from simple commonsense notions of plan relations and interactions. It would be valuable to define a formal semantic model that grounds the qualitative relations in abstractions of causal-link plan structures. Such a model would enable us to show that the calculus ‘does the right thing’, rather than simply matching intuitions as to what should happen.

Acknowledgments This work was supported by DAPRA under Air Force Research laboratory contract F30602-00-C-0058. The author would like to thank Peter Jarvis, Mabry Tyson, and Michael Wolverton for their assistance in clarifying the technical ideas in this work.

7 Bibliography

Allen, J. and Ferguson, G. (2002). “Human-Machine Collaborative Planning”. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Carroll, J. M. and Moran, T. P. (1991). Special issue on design rationale. *Human-Computer Interaction*, 6(3-4).

Conklin, E. J. and Yakemovic, K. B. (1991). "A Process-oriented Approach to Design Rationale". *Human-Computer Interaction*, 6:357--391.

El Fattah, Y. (2003). "Heuristic Causal Modelling for Mixed Initiative Planning". Submitted for Publication.

Erol, K., Hendler, J., and Nau, D. (1994). Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Kambhampati, S. and Hendler, J. (1992). "A Validation-structure Based Theory of Plan Modification and Reuse", *Artificial Intellgience*, 53(2), 193-258..

Kim, J. and Blythe, J. (2003). "Supporting Plan Authoring and Analysis", In *Proceedings of the Seventh International Conference on Intelligent User Interfaces (IUI-2003)*.

McCarthy, J. Epistemological problems of Artificial Intelligence. (1977). In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77).*

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., and desJardins, M. (2002). "PASSAT: A User-centric Planning Framework". In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Myers, K. L., Jarvis, P. Tyson, W. M., and Wolverton, M. J. "A Mixed-initiative Framework for Robust Plan Sketching". (2003). In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, Trento, Italy.

Weld, D. (1999). "Recent Advances in AI Planning", *AI Magazine*, 20(2), 93-123.

Active Coordination of Distributed Human Planners

Karen L. Myers Peter A. Jarvis Thomas J. Lee
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.,
Menlo Park, CA 94025
{myers, jarvis, tomlee}@ai.sri.com

Abstract

Effective coordination of distributed human planners requires timely communication of relevant information to ensure the overall coherence of activities and the compatibility of assumptions. This paper presents a framework called CODA that provides targeted information dissemination among distributed human planners as a way of improving coordination. Within CODA, each planner declares interest in different types of plan change that could impact his or her local plan development. As individuals develop plans using a plan authoring tool, their activities are monitored; changes that match declared interests trigger automatic notification of appropriate planners. In this way, distributed planners can receive focused, real-time updates of plan changes that are relevant to their local planning efforts.

Introduction

The scope and complexity of large-scale planning tasks generally requires cooperation among multiple planners with differing areas of expertise, each of whom contributes portions of the overall plan. These planners may be distributed both geographically and temporally, further complicating coordination.

As a concrete illustration, special operations forces (SOF) mission planning involves numerous people working on separate but interconnected facets (e.g., strategic, logistical, medical, intel) of an overall plan. The SOF planning process itself is time constrained, concurrent, and iterative. Individual planners construct subplans based on their expectations for the operating environment and requirements. As the overall plan develops, these expectations change and modifications must be made to reflect new information. Currently, such changes are communicated informally by word of mouth, or transmitted in batch mode at regularly scheduled coordination sessions. This approach can lead to omissions and delays that reduce the effectiveness of the overall planning process and the quality of the resulting plans.

The SOF planning domain lies well beyond the range of current automated planning technologies. Moreover, fully automated solutions are unlikely ever to succeed, due to two main factors. First, effective planning for this domain involves a huge strategic component that requires evaluating options with respect to expected plan quality. This type of knowledge is extremely difficult to capture and model, being grounded in a combination of intuition, vast experience, and common sense. Planning knowledge bases constructed to date mostly ignore strategic issues, focusing instead on modeling preconditions/postconditions for action applicability and effects. Second, SOF planning tasks (e.g., disaster relief, counterterrorism) tend to be unique and distinctive, making it difficult to formulate reusable background knowledge with adequate coverage. As an additional consideration, human planners in this area are reluctant to cede full control to automated planning systems, even in situations where automation is possible.

The above characteristics apply to a range of planning domains, including more general military operations planning, workflow management, and certain types of space missions. Techniques from the AI planning community can still contribute to complex problem solving in these domains. In particular, *plan authoring* tools that build on AI planning concepts are being introduced to improve the quality and process of plan development (Valente *et al.* 1999; GTE 2000; Muñoz-Avila *et al.* 1999; Knoblock *et al.* 2001; desJardins *et al.* 2002). Plan authoring tools provide a set of plan editing operations and manipulation capabilities that support users in exploring and developing plans. These tools may provide some automated capabilities; however, their main role is to augment rather than replace human planning skills. Plan authoring tools introduce a degree of structure to the planning process, yielding principled representations of plans with well-defined semantics. Planning aids that reason over these structures can be defined, including tools to support interplanner coordination.

Three main considerations drive the development of coordination techniques within this type of setting.

Selectivity Coordination strategies must strike a balance between disseminating too little and too much information. Failure to communicate enough information could be disastrous, if planners do not receive notification of critical changes. Conversely, flooding planners with much extraneous information can lead to cognitive overload, with critical changes being lost in a sea of irrelevant updates. Overcommunication can be a further problem for

situations where bandwidth is limited (e.g., wireless devices) or communication is expensive.

Timeliness Information about plan changes must be received in a timely manner to ensure adequate time for human planners to assess impact on their local plans and develop appropriate repairs.

Nonintrusiveness Users are generally reluctant to embrace technologies that require changes to traditional work habits (Conklin & Yakemovic 1991). In particular, a user is unlikely to perform activities beyond his normal sphere of responsibility unless there are immediate and substantial benefits for him. Thus, coordination tools that impose demands on users must incentivize their participation.

This paper describes the CODA (Coordination of Distributed Activities) framework, which provides automated support for focused information sharing during collaborative plan development by a team of humans. While motivated by the SOF planning problem, CODA more generally targets applications where distributed human planners are assigned responsibility for developing subportions of a global plan. These subplans are expected to have a moderate degree of coupling due to the need for coherent strategy, coordinated actions, and sharing of limited resources.

Within CODA, each planner declares the kinds of plan change that are of interest to him or her; we call these declarations *plan awareness requirements* (or *PARs*). As users develop plans with a plan authoring tool, their activities are monitored. Changes that match plan awareness requirements are forwarded automatically to the person who declared interest in them. In this way, distributed planners can receive focused, real-time updates of plan changes that are relevant to their local planning efforts.

Because local planners declare precisely the information in which they are interested, CODA satisfies the criteria for selectivity stated above. The declaration of plan awareness requirements constitutes the only additional effort required by the human planners above and beyond their normal operation. Because this declaration process will produce direct benefits to the planner (namely, notification of changes in which they are interested), the motivation for this specification phase is strong. Finally, the real-time nature of the plan updates within CODA addresses the timeliness concern.

The paper begins with an overview of the planning model within CODA, followed by a description of the CODA architecture. Next, we define our language for expressing plan awareness requirements along with a formal semantics and algorithms for matching them to evolving plans. We then describe an implementation of the CODA framework, focusing on different matching modes that it supports and tools for creating and registering plan awareness requirements. Finally, we compare our approach to related work in distributed AI, active databases, and concurrent engineering.

Plan Model

The generative planning community typically models a plan as a partial order of actions, and planning as an action selection problem. More generally, planning can be viewed as a *design task* that involves creating, refining, and linking objects to produce a composite structure that satisfies stated design objectives. The plan authoring model underlying CODA embodies this more general model of planning.

Objective a goal to be achieved within a plan

Action an activity that can be performed to achieve objectives

Effect a world-state condition denoting an expected result of plan activities

Event a world-state condition denoting an expected externality

Decision Point a condition for branching among subplans

Role a required capacity within a plan

Relation a semantic connection among plan objects (e.g., an Action *supports* an Objective or *enables* an effect)

Figure 1: Plan Ontology for CODA

Plan Ontology and Structure A CODA plan is composed of a collection of objects drawn from the plan ontology presented in Figure 1. Each plan object is characterized by a unique *id* and may optionally have a (not necessarily unique) *name*. A collection of *domain objects* augments the plan ontology, representing the basic entities within a specific domain.

While most of the ontology elements correspond to standard concepts in AI planning, *roles* are somewhat different. A role defines an abstract capacity within a plan, independent of the specific object that will eventually fill that capacity. For example, *place* roles are used frequently within the SOF domain: in a disaster relief plan, there may be one or more *assembly-point* roles that denote places where evacuees should assemble to be evacuated. A planner often knows early on that assembly-points with certain capacities are required, but the exact physical locations for them and the actions that make use of them may not be chosen until later in the planning process.

Planning Process The planning process for CODA involves specifying roles required within a plan, selecting objects to instantiate those roles, selecting particular actions to be executed, declaring expected effects and events, and asserting relationships among objects and actions. The planning process is incremental in nature, involving the selection, extension, and refinement of selected objects and actions until the plan meets the human planner's criteria for adequacy with respect to stated objectives.

Plan Query Language The language for expressing plan awareness requirements (described in a subsequent section) builds on a general-purpose plan query language for the CODA plan ontology. The query language consists of a typed first-order language specialized to the Generic Frame Protocol (GFP) model of frame representation systems (i.e., it includes the full range of GFP-defined functions and predicates for manipulating classes, instances, and relationships

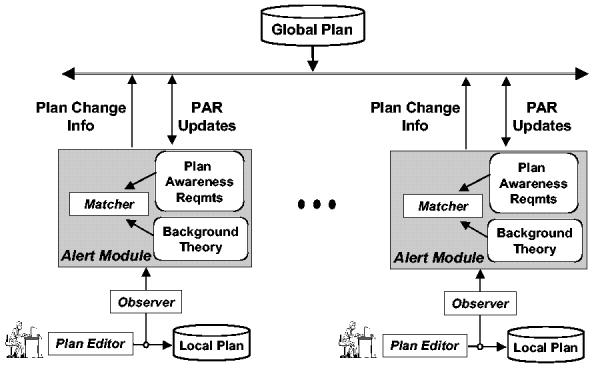

Figure 2: CODA Architecture

(Karp, Myers, & Gruber 1995)). Thus, for example, the language supports predicates of the form `(C <x>)` and terms of the form `(Attr <x>)` for every class `C` and attribute `Attr` defined within the ontology. The language further includes equality, term constructors for lists and intensionally defined sets, and quantification with respect to an enumerable type.

CODA Architecture

Figure 2 presents the architecture of CODA. Within the context of a global plan, individuals work independently to produce local plans for their assigned tasks. Plans are developed using a structured *plan editor*, which supports a broad range of plan manipulation capabilities. User interactions with the plan editor are tracked by an *observer* module, which maintains a complete history of editing operations. As events are logged, a semantically grounded representation of the local plan is built within CODA. This internal representation can be annotated and used for reasoning, independent of the plan editor GUI.

The *matcher* provides the main inferential capability within CODA, being responsible for linking observed plan changes to declared PARs. The matching process may involve reasoning with a *background theory*, whose role is to bridge the gap between low-level plan edits and PARs expressed in high-level languages. When matches are detected, notification is sent to the local planner who registered the matched plan awareness requirement.

SOFTools Temporal Plan Editor CODA could be linked to a variety of manual and automated planning tools. Currently, it is connected to a specific plan editor, the SOFTools Temporal Plan Editor, which was developed to support graphical editing of SOF plans (GTE 2000). SOF plans involve the coordination of large numbers of activities and resources, subject to complex temporal synchronization constraints. CODA's event monitoring for the Temporal Plan Editor covers most of the available editing operations, including creation, modification and deletion of objects, modification of object attributes, temporal shifting of activities, and resource assignment.

Background Theory The background theory extends the inferential capabilities of the matching process by allowing definition of PARs in a high-level language whose expressivity extends beyond the basic plan query language. As such, the background theory enables a greater separation between observable plan modifications and the vocabulary for expressing PARs.

The background theory encompasses three types of information. The first specializes the CODA plan ontology with domain-specific information. For example, each domain would have its own specialization of the plan ontology class *Event*, which could be organized into a hierarchical structure of subclasses (e.g., *Weather-Event*, *Equipment-Event*) and instances (e.g., *Hurricane*). The second type consists of the domain objects, such as locations and resources, which are also structured in hierarchical fashion. Our CODA system stores these two types of background information within the Ocelot frame representation system (Karp, Chaudhri, & Paley 1999). The third type extends the underlying plan language with functions and relations defined over plan and domain objects. For example, our SOF application of CODA includes functions for computing distance and compass bearing between geographical points, and capacity analysis functions.

To appreciate the value of the background theory, consider the class of plan changes corresponding to *The addition of insertion points within two miles of an existing insertion point*. Our representation of plans supports the creation and manipulation of specific types of insertion points (e.g., `Helicopter-Landing-Zones`, `Drop-Points`) with a range of attributes that includes geographical location. The background theory in this case provides the ability to refer to a collection of instances (i.e., the elements of the type `Insertion-Point`), as well as to measure geographic distance between objects. The above plan change would be cumbersome to describe without these predefined background concepts.

Plan Awareness Requirements

We define two types of plan awareness requirement: *plan-state* and *transition*. Plan-state PARs describe conditions of a plan; in contrast, transition PARs describe *changes* to a plan.

Plan-State PARs

A plan-state PAR describes conditions of a plan and is modeled in terms of a well-formed formula in the plan query language. For example, a plan-state PAR for

There is an arrival to FSB Gold scheduled for after 8 AM

would be represented by the following formula in the plan query language[1]:

```
(EXISTS (?X ?Y)
   (AND (MOVE ?X) (FSB ?Y)
        (= (NAME ?Y) GOLD))
        (= (DESTINATION ?X) ?Y)
        (> (ARRIVAL-TIME ?X) 800))
```

[1] Symbols preceded by `?` (e.g., `?x`) denote variables.

Matching of a plan-state PAR occurs when a modification yields a plan that satisfies the associated plan query.

Plan Transition PARs

In its most general form, a plan transition PAR describes two plan states P and P', and changes that transform the former into the latter. Transition PARs are grounded in the modification of individual or groups of objects. For this reason, transition PARs are defined in terms of an *object schema* or a *set specification*.

An object schema denotes a plan object with designated characteristics. Object schemas are specified using the syntax $(ANY\ ?x\ \phi[?x])$ where ϕ is an arbitrary formula in the plan query language. A set specification, given by the syntax $(SET\ ?x\ \phi[?x])$, designates a collection of objects with specified characteristics. We use the symbol σ to denote object schemas and the symbol Φ to denote set specifications.

We distinguish several categories of transition PAR, based on the nature of the underlying plan changes. These categories correspond to the creation, deletion or modification of plan objects, the refinement or generalization of plan object attributes, changes to specific attributes of a plan object, and changes to a collection of plan objects.

Instance Creation Instance Creation PARs declare interest in the addition of an object to a plan that satisfies stated conditions. An Instance Creation PAR is defined by an object schema σ that describes characteristics of the plan object to be created. For example, interest in

Addition of decision points related to weather calls

would be represented by an Instance Creation PAR with object schema

```
(ANY ?X (AND (DECISION-POINT ?X)
             (= (TYPE ?X) Weather-Call)))
```

Instance Deletion Instance Deletion PARs declare interest in the removal of an object from a plan that satisfies stated conditions. An Instance Deletion PAR is defined by an object schema that describes characteristics of the plan object to be deleted. For example, interest in

Elimination of a landing zone south of the embassy

would be represented by an Instance Deletion PAR with object schema

```
(ANY ?x (AND (LANDING-ZONE ?x)
         (= South
            (DIR (POSN ?x)(POSN Embassy)))))
```

Instance Modification This class of PARs declares interest in the modification of an object that satisfies stated conditions. An Instance Modification PAR is defined by an object schema that describes the plan objects to be modified. As an example, interest in

Changes to 4th-platoon movements

would be represented by an Instance Modification PAR with object schema

```
(ANY ?X (AND (MOVE ?X)
          (= 4th-platoon (OPERAND ?X))))
```

Attribute Refinement Attribute Refinement PARs declare interest in the assignment of values to unbound attributes of plan objects. They are defined as a pair $\langle\sigma,A\rangle$ where σ describes a class of plan objects, and A the attribute to be bound.

Attribute Decommitment Similar to Attribute Refinement PARs, Attribute Decommitment PARs declare interest in decommitment from assigned attribute values. They are defined as a pair $\langle\sigma,A\rangle$ where σ describes a class of plan objects, and A the attribute to be decommitted.

Attribute Modification This class of PARs specializes Instance Modification PARs to changes to a specific attribute of a plan object. PARs of this type are defined as a triple $\langle\sigma,A,\delta\rangle$, where σ describes a class of plan objects, A the attribute of interest, and δ an optional *change predicate* that imposes constraints on the nature of the change. The change predicate can restrict the new value (e.g., `(= ?NEW 5)`) or the relationship between the old and new values (e.g., `(> (- ?NEW ?OLD) 3)`). For example, interest in

Delays of > 1 hour in expected time to secure the Church

would be captured by an Attribute Modification PAR $\langle\sigma,A,\delta\rangle$ with the components

```
σ   (ANY ?X (AND (EFFECT ?X)
                 (= (TYPE ?X) Secure)
                 (= (OPERAND ?X) Church)))
A   TIME
δ   (> (- ?NEW ?OLD) 1)
```

Aggregate Modification This class of PARs can be used to declare interest in changes to an intensionally defined collection of objects. The change may be to membership in the collection, or to some *aggregation value* defined over the collection. For example, an aggregation could be defined in terms of the movements that involve a particular type of personnel, with the aggregation value defined as the sum of resources employed by those movements.

An Aggregate Modification PAR is defined as a 4-tuple $\langle\Phi,A,\pi,\delta\rangle$, where Φ describes a set of plan objects, A the attribute to change, and δ a change predicate. The *aggregation function* π reduces a set of values to a single value; common aggregation functions include *MIN*, *MAX*, *SUM*, *COUNT*, and *AVERAGE*. As an example, the change

Decrease of more than 2 in the number of UH-60s used

would be represented by an aggregate modification PAR $\langle\Phi,A,\pi,\delta\rangle$ defined as follows:

```
Φ   (setof ?X (AND (MOVE ?X)
                   (= UH-60 (ASSETTYPE ?X))))
A   ASSETCOUNT
π   SUM
δ   (>= (- ?NEW ?OLD) 2)
```

The change predicate, attribute specification, and aggregation functions within Aggregate Modification PARs are each optional. Omission of the change predicate indicates that any change is acceptable. Omission of the attribute indicates that the aggregation function should be applied directly to the objects in the set designated by Φ. Omission

of the aggregation function indicates interest in the composition of the set of objects (or their attribute A); this last case corresponds to a 'modify set' semantics.

Match Semantics

We use the notation $\langle E,P,P'\rangle$ to denote a set of plan edit operations $E = \{e_1,\ldots\ e_m\}$ that maps plan P into a revised plan P'. The notation $Holds(\phi)$ indicates that the plan-state formula ϕ is a logical consequence of the background theory, while $Holds(\phi,P)$ indicates that ϕ is a logical consequence of plan P conjoined with the background theory. Finally, $k \in P$ ($k \notin P$) indicates that the plan object k is defined (is not defined) within plan P.

First, we define the concept of *domain* for an *object schema* and a *set specification.*

Definition 1 (Domain: Object Schema, Set Specification) *The* domain *of an object schema* $\sigma = (ANY\ ?x\ \phi[?x])$ *or set specification* $\Phi = (SETOF\ ?x\ \phi[?x])$ *for a plan P, denoted by* $Domain(\sigma,P)$ *and* $Domain(\Phi,P)$*, respectively, consists of the set of plan objects* $\{k_1\ldots\ k_n\}$ *within P for which* $Holds(\phi[k_i],P)$.

Definitions 2 and 3 capture the semantics for PAR matching.

Definition 2 (Match of Plan State PAR) *A plan state PAR* $\langle\phi\rangle$ *matches* $\langle E,P,P'\rangle$ *iff* $Holds(\phi,P')$.

Definition 3 (Match of Transition PAR) *A transition PAR matches* $\langle E,P,P'\rangle$ *under the following conditions:*

Instance Creation $\langle\sigma\rangle$*: There exists a plan object k such that* $k \notin P$ *but* $k \in Domain(\sigma,P')$.

Instance Deletion $\langle\sigma\rangle$*: There exists a plan object k such that* $k \in Domain(\sigma,P)$ *but* $k \notin P'$.

Instance Modification $\langle\sigma\rangle$*: There exists a plan object k and attribute A of k such that*

- $k \in Domain(\sigma,P)$, $k \in P'$
- $Holds(ATTR(k,A) = v,P)$
- $Holds(ATTR(k,A) = v',P')$
- $Holds(v \neq v')$

Attribute Refinement $\langle\sigma,A\rangle$*: There exists a plan object k such that*

- $k \in Domain(\sigma,P)$, $k \in P'$
- $Holds(ATTR(k,A) = v',P')$
- *there is no v for which* $Holds(ATTR(k,A) = v,P)$

Attribute Decommitment $\langle\sigma,A\rangle$*: There exists a plan object k such that*

- $k \in Domain(\sigma,P)$, $k \in P'$
- $Holds(ATTR(k,A) = v,P)$
- *there is no v' for which* $Holds(ATTR(k,A) = v',P')$

Attribute Modification $\langle\sigma,A,\delta\rangle$*: There exists a plan object k such that*

- $k \in Domain(\sigma,P)$, $k \in P'$
- $Holds(ATTR(k,A) = v,P)$
- $Holds(ATTR(k,A) = v',P')$
- $Holds(v \neq v')$
- *if* $\delta \neq \emptyset$ *then* $Holds(\delta[v,v'])$

Aggregate Modification $\langle\Phi,A,\pi,\delta\rangle$*: Define* π^* *to be the identify function when* $\pi = \emptyset$ *and* π *otherwise. Let* $D = Domain(\Phi,P)$ *and* $D' = Domain(\Phi,P')$*. For v and v' defined by*

$$v = \begin{cases} \pi^*(\{ATTR(x,A) \mid x \in D\}) & if\ A \neq \emptyset \\ \pi^*(D) & if\ A = \emptyset \end{cases}$$

$$v' = \begin{cases} \pi^*(\{ATTR(x',A) \mid x' \in D'\}) & if\ A \neq \emptyset \\ \pi^*(D') & if\ A = \emptyset \end{cases}$$

it is the case that

- $Holds(v \neq v')$
- *if* $\delta \neq \emptyset$ *then* $Holds(\delta[v,v'])$

The semantics for matching an Instance Creation PAR embody a *strict* interpretation of creation that requires the generation of a new object within the domain of the PAR's object schema σ. A more liberal interpretation would match a plan transition whose changes move an existing object from outside to inside the domain of σ. For example, changing the type of a decision-point from `equipment-check` to `weather-call` could be interpreted as matching an Instance Creation PAR with object schema

```
(ANY ?X (AND (DECISION-POINT ?X)
             (= (TYPE ?X) weather-call)))
```

Similarly for Instance Deletion PARs, one might consider matches to include transitions where an object was modified so that it no longer satisfies the conditions of σ.

We opted for the stricter interpretation of change for two reasons. First, it matches more closely intuitions for creation and deletion. Second, changes that correspond to the more liberal interpretation can be expressed as Aggregate Modification PARs. For example, the more liberal interpretation of an Instance Creation PAR with object schema $\sigma = (ANY\ ?x\ \phi[?x])$ can be expressed by the Aggregate Modification PAR

$$\langle (SETOF\ ?x\ \phi[?x]), \emptyset, \texttt{SUM}, \texttt{(> ?NEW ?OLD)} \rangle$$

which captures the notion of an increase in the number of objects that satisfy $\phi[?x]$.

Matching Algorithms

The matching of a plan state PAR reduces to the performance of a plan query. Matching of transition PARs is somewhat more complex.

The transition PARs that focus on instances and attributes (namely, Instance Creation/Deletion/Modification and Attribute Refinement/Decommitment/Modification PARs) have a decomposable nature. Testing for matches to these PARs reduces to checking for changes to plan objects within individual plans. As such, these PARs can simply be broadcast to the distributed CODA modules, with matching performed locally and results returned independently to the CODA module that registered the PAR.

Aggregate Modification PARs are not decomposable in this manner because they embody an implicit quantification over the union of the local plans under development. Testing of these PARs requires reasoning about changes across all

subplans. To illustrate, consider the PAR that represents the change

Decrease of more than 2 in the number of UH-60s used

If three individual planners each decrease their usage of UH-60s by one, their collective change results in a match to the PAR. Detecting a match of this type involves reasoning at an aggregate level about changes that have been made by the team of distributed planners.

Fortunately, the matching of Aggregate Modification PARs can be reduced to a combination of local matching and plan querying. For a given Aggregate Modification PAR

$$\gamma = \langle (SETOF\ ?x\ \phi[?x]), A, \pi, \delta \rangle$$

define the following *derived* PARs:

- *Attribute Modification PAR* $\gamma^{AM} = \langle (ANY\ ?x\ \phi[?x]), A, \emptyset \rangle$
- *Instance Creation PAR* $\gamma^{IC} = \langle (ANY\ ?x\ \phi[?x]) \rangle$
- *Instance Deletion PAR* $\gamma^{ID} = \langle (ANY\ ?x\ \phi[?x]) \rangle$

A match to any of the derived PARs in $\Gamma = \{\gamma^{AM}, \gamma^{IC}, \gamma^{ID}\}$ constitutes a necessary but not sufficient condition for match of γ: for γ to match, there must either be a change to one of the objects in the set (captured by γ^{AM}), or to membership in the set (captured by the combination of γ^{IC} and γ^{ID}).

Upon notification of a match to a PAR in Γ, the CODA module that registered γ can confirm a match for γ by aggregating the results from a set of distributed queries to each CODA module L_i. These queries consist of

$$Objects_{L_i} = Query[(SETOF\ ?x\ \phi[?x])]$$

$$Values_{L_i} = \{\ Query[ATTR(?x, A)]\ \mid\ ?x \in Objects_{L_i}\}$$

for each local planner L_i. Determination of a match then involves evaluating the change predicate δ on the result returned by the application of the aggregation function to the union of these values, that is:

$$\delta(\pi(\cup_{L_i} Values_{L_i}))\ .$$

CODA System

Figure 3 shows the interface for the CODA system (on the left) beside that of the SOFTools Temporal Plan Editor (on the right). Within the graphical plan representation of the Temporal Plan Editor, labeled horizontal lines correspond to places (e.g., a staging base or the Hospital), arrows correspond to actions, and diamonds denote key effects (e.g., the securing of a building or the attainment of a position).

The interface in the figure corresponds to a state in which a human is developing the maneuver portion of a plan to rescue hostages from a town held by enemy forces. Working from the top downward, the plan consists of two assault teams flying by helicopter from a warship to the Gold and Silver staging bases just outside the town. The assault teams then move by foot to the town and secure the buildings in which the hostages are being held (the Church, Hospital, and Town-Hall). The hostages are collected and taken to a third staging base (Bronze) from which they are flown to safety in a transport aircraft.

Another human planner, whose interface is not depicted, is developing the fire-support portion of the same plan. The bottom half of CODA's interface displays the PARs registered by these two planners. The window labeled *Own PARs* indicates that the maneuver planner is interested in any delay of more than 15 minutes to the planned time of fire-support assets arriving on station. The window labeled *Others' PARs* shows the fire-support planner's interest in changes to the planned locations of helicopter landing zones (HLZs). The *Matches* window informs the maneuver planner that the fire-support planner has changed his plan in such a way as to delay the time at which a fire-support asset will be on station.

Matching Modes

The CODA system supports two modes for registering PARs, based on the frequency with which users are notified of matches.

PARs registered in *immediate* mode are checked after every plan edit operation (i.e., $E = \{e\}$ in Definition 3), thus providing planners with real-time notification of relevant plan changes. Immediate notification would be suitable for the endstages of planning (when plans are mostly stable and changes are significant), or during execution.

For earlier stages of plan development, frequent and wide-ranging changes to plans would be expected; real-time notification of matches during early plan development would be counterproductive. For PARs registered in *on-demand* mode, matching information is provided only in response to an explicit user request. Such requests produce summaries of matches for the current plan relative to a designated 'checkpoint' plan. On-demand matching can support coordination of distributed planners earlier in the planning process by enabling a given planner to periodically check for changes by other planners that could impact his own efforts.

PAR Definition and Registration

Within the CODA system, PARs can be registered and unregistered dynamically throughout a planning session. This flexibility is important because the informational needs of planners will necessarily evolve in response to changes in guidance, the dynamics of the environment, and the unfolding of the planning process itself. For example, a planner may wish to be notified in the event that spare capacity is available to transport certain extra cargo that would be required for one option under consideration; if he decides to adopt a different strategy that does not require the cargo, then he would unregister interest in available capacity.

PARs could originate from a variety of sources. Within CODA currently, PARs are either (a) selected from predefined libraries, or (b) created by individual planners on an as-needed basis.

PAR Libraries Libraries group PARs for standard types of related plan change that would generally be of interest to planners within a given domain. For example, a team responsible for medical needs during an evacuation would

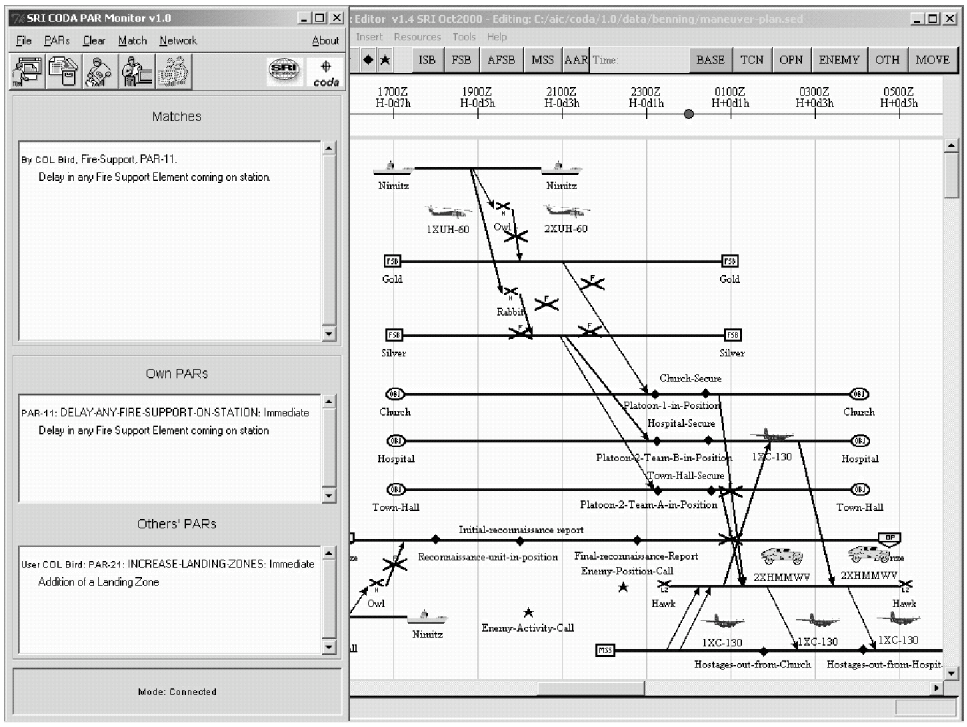

Figure 3: Interfaces for CODA (left) and the Temporal Plan Editor (right)

always be interested in changes to the size of the force and the expected number of evacuees. PARs for these types of change can be stored in a library specialized to the needs of medical planners. During a particular planning session, a planner would draw on libraries of predefined PARs to form the basis of his registered interests, augmenting them as necessary with authored PARs tied to the situation at hand and the current plan.

PAR Authoring CODA includes an interactive *PAR authoring tool* designed to help users quickly define PARs not contained within predefined libraries. This tool was developed using Adaptive Forms (Frank & Szekely 1998), a grammar-based framework that supports the specification of structured data through a form-filling interface that adapts in response to user inputs. With this tool, users create PARs by filling in forms with an English-like syntax; as users incrementally specify PARs, remaining options change in accord with the constraints of the underlying grammar. An internal compiler transforms these high-level specifications into the formal PAR structures required by CODA's matcher.

In designing a specification tool of this type, the competing requirements of expressivity and ease of use must be balanced. Sufficient expressivity is required to ensure coverage of relevant cases; however, support for full expressivity can lead to complex and unintuitive interfaces. To address this issue, CODA's PAR authoring tool provides two sets of forms. First, a set of *general forms* provides the full expressive power of the PAR language, including the ability to construct arbitrary expressions in first-order logic. While powerful, these forms require more effort to complete; in addition, people unaccustomed to formal languages require training to use them effectively.

For this reason, the tool also includes specializations of the general forms that capture common *idioms* within the SOF planning domain. These specialized forms build in values that users would have to specify in the general case, thus simplifying and shortening the specification process. Parameters within the forms enable customization to a given planning session. SOF planners, for example, are generally interested in delays to activities. The SOF application of CODA includes (among others) the following parameterized PAR idiom:

Delays to any actions of greater than < duration >
Delays to action < action-id >

The first form supports declaration of interest in delays to actions that exceed a duration to be supplied by the user. The second form supports declaration of interest in delays to a user-specified action within a plan. Users can create PARs based on these specialized forms simply by supplying the designated parameters.

The idiomatic forms serve as the primary mechanism for authoring PARs within the CODA system, with the general-purpose forms reserved for defining PARs that lie beyond the scope of the predefined idioms.

Related Work

Distributed AI

Several AI planning systems have been built that provide coordination mechanisms to share information among dis-

tributed planners. In contrast to CODA, these systems link automated planners that construct plans with comprehensive causal structures. Analysis of plan dependencies within these structures forms the basis for determining the information to share among planners.

The COLLAGE planner (Lansky 1998) supports the conceptual partitioning of a planning problem into collections of regions, each of which constitutes a planning problem in its own right. Constraint propagation rules are used to maintain consistency among regions.

In the DSIPE distributed planning framework (Wolverton & desJardins 1998), each planner publishes a list of predicates that are *relevant* to its planning needs. Relevance is determined automatically through a reachability analysis of goals and operators. As planning proceeds, a planner determines whether a given planning decision establishes effects that unify with published relevant predicates from other planners, and notifies them as appropriate.

DSIPE's approach can be viewed as a counterpart to the CODA framework in which PARs are generated automatically. Because this automation relies on a complete and comprehensive set of planning operators, its applicability to plan authoring systems (where such information will not be available) is limited. Furthermore, the PAR representation provides a richer language for describing plan changes, going well beyond the simple effects-oriented approach within DSIPE.

Active Databases

Active databases augment traditional database technology with a set of rules that trigger activities in response to database modifications (Widom & Ceri 1996). Rules have the form $\langle E,C,A\rangle$ where E is an event that triggers invocation of the rule, C is a condition to be satisfied, and A is an action to be performed when C is met.

Although parallels can be drawn between PARs and active database rules, several characteristics distinguish our work. First, our approach involves monitoring changes to plans, which have richer structure than do relational or object-oriented databases. Second, we emphasize user-friendly languages and tools for declaring PARs, while the active database community has focused on automated synthesis of rules to support tasks such as the enforcement of integrity constraints. Finally, active database work does not consider the incorporation of background theories into the matching process.

Concurrent Design

The concurrent design community has developed techniques for detecting and resolving conflicts that arise during large-scale distributed design tasks (Klein 1993; Petri 1993). The general approach involves the documentation of *design rationale* – an explicit representation of design decisions and their interdependencies. As changes are made, the captured rationale supports automatic identification of decisions that might be impacted and should be reconsidered. The design rationale must be specified by users, thus imposing a substantial documentation burden during the design process. In contrast, our approach avoids such onerous documentation requirements but leaves the task of ascertaining the ramifications of PAR matches to the user.

Conclusions

There has been a recent trend within the field of AI planning to increase relevance by focusing on more realistic and challenging problems drawn from real-world applications. Deeper exploration of several motivating domains (e.g., military operations planning, workflow management, space mission science planning) has revealed the impracticality of fully automated systems for many of these planning tasks. In particular, formulation of the knowledge bases required to support generation of high-quality plans in these domains lies well beyond the reach of current and forseeable modeling capabilities. Instead, the experience, common sense, and intuitions of the human decision-maker are essential for effective planning in these domains. For this reason, many researchers within the AI planning community have begun developing *plan authoring tools* designed to assist a human decision-maker in constructing high-quality plans rather than to replace him.

The CODA framework provides a practical solution to the problem of coordinating the activities of distributed human planners engaged with such plan authoring tools. By having human planners explicitly declare those aspects of the overall planning process that interest them, CODA enables timely and focused distribution of information that can expedite and improve the quality of coordinated problem solving. The use of a rich, AI-based representation for describing plans, planning changes, and background theories provides the key to this technology.

CODA is a general-purpose coordination framework that can be linked to a variety of plan authoring tools. However, its development has been strongly influenced by the real-world challenges of SOF mission planning. Several domain experts provided guidance in the formulation of CODA to ensure that it meets the expressivity and usability needs of potential users. In addition to our current layering of the CODA system on top of the Temporal Plan Editor, we are looking to link CODA to additional types of plan authoring systems, such as an asset allocator and a map-based planner. Linkage of these tools through CODA will enable a rich and heterogeneous distributed plan authoring environment that provides greatly improved coordination over current practice.

Acknowledgments This work was supported by DARPA under Air Force Research Laboratory contracts F30602-97-C-0067 and F30602-00-C-0058. The authors would like to thank Fred Bobbitt, Doug Dyer, Warren Knouff, and Kelly Snapp for their help in shaping the development of CODA.

References

Conklin, E. J., and Yakemovic, K. B. 1991. A process-oriented approach to design rationale. *Human-Computer Interaction* 6:357–391.

desJardins, M.; Myers, K. L.; Tyson, M.; Wolverton, M.; Jarvis, P.; and Lee, T. 2002. PASSAT: From automated

planning to plan authoring. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA.

Frank, M. R., and Szekely, P. 1998. Adaptive Forms: An interaction paradigm for entering structured data. In *Proceedings of the ACM International Conference on Intelligent User Interfaces*, 153–160.

GTE. 2000. *SOFTools User Manual.*

Karp, P. D.; Chaudhri, V. K.; and Paley, S. M. 1999. A collaborative environment for authoring large knowledge bases. *Journal of Intelligent Information Systems* 13:155–194.

Karp, P. D.; Myers, K. L.; and Gruber, T. 1995. The Generic Frame Protocol. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence.*

Klein, M. 1993. Supporting conflict management in cooperative design teams. *Group Decision and Negotiation* 2:259–278.

Knoblock, C. A.; Minton, S.; Ambite, J. L.; Muslea, M.; Oh, J.; and Frank, M. 2001. Mixed-initiative, multi-source information assistants. International World Wide Web Conference.

Lansky, A. L. 1998. Localized planning with action-based constraints. *Artificial Intelligence* 98(1-2):49–136.

Muñoz-Avila, H.; Aha, D. W.; Breslow, L.; and Nau, D. 1999. HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations. In *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence*, 879–885. AAAI Press.

Petri, C. 1993. The Redux' server. In *Proceedings of the International Conference on Intelligent and Cooperative Information Systems (ICICIS).*

Valente, A.; Blythe, J.; Gil, Y.; and Swartout, W. 1999. On the role of humans in enterprise control systems: The experience of INSPECT. In *Proceedings of the DARPA-JFACC Symposium on Advances in Enterprise Control.*

Widom, J., and Ceri, S., eds. 1996. *Active Database Systems.* Morgan Kaufmann.

Wolverton, M. J., and desJardins, M. 1998. Controlling communication in distributed planning using irrelevance reasoning. In *Fifteenth National Conference on Artificial Intelligence.* AAAI Press/MIT Press.